Unlinkable Divisible Digital Cash without Trusted Third Party *

Pawel Pszona and Grzegorz Stachowiak, University of Wroclaw, Poland

July 5, 2007

Abstract

We present an efficient divisible digital cash scheme which is unlinkable and does not require Trusted Third Party. The size of the coin is proportional to the size of the primes we use, i.e., hundreds of bytes. The computational and communication complexity of the protocol is proportional to a polynomial of the size of the primes and polylogarithm of the maximum number of pieces to which a coin can be subdivided.

Keywords: digital cash, divisibility, unlinkability

1 Introduction

Electronic cash has drawn much attention since it was introduced by Chaum [8]. Its basic idea is to provide an electronic payment scheme that ensures that the payer stays anonymous to the merchant during a transaction, yet the payer isn't able to cheat the merchant by spending false (forged) coins.

Anonymity should also hold between a user and a bank (even though the bank issues the coin, it shouldn't be able to determine which user a payment comes from, given the transcript of this payment). Additionally, in order to prevent the merchant from accepting a false coin, the scheme has to force the user to prove in a sound fashion that she uses a valid coin.

The security of real cash depends on the difficulty of reproducing it. Since this is not the case with electronic cash and a user may want to spend more than she is allowed to (this is called *double-spending*), the system has to guarantee that this situation is detected and the identity of a cheating user is disclosed.

There are two major kinds of electronic cash systems – *on-line* schemes (e.g. [15], [3]), where the bank participates in each payment between a user and a merchant, and *off-line* schemes, where a merchant is on his own as to decide whether to accept a payment or not. After the payment is executed, the merchant presents the bank with a transcript of that payment and this payment's value is added to his bank account. Most proposed schemes are off-line, as they seem more suitable for real world applications.

It would be nice to have a system where a coin could be transferred between users many times before being deposited to a bank. However, as shown in [11], a transferred coin would have to grow in size (to store all of its spending history), worsening the scheme's efficiency. Therefore most proposed schemes don't support transferability.

A problem arises with the value of a payment. Many proposed schemes (e.g. [10], [2]) are *exact* schemes. This means that a coin may be spent only using all of its value in a single

*A very similar result was independently obtained by Sébastien Canard and Aline Gouget and published on Eurocrypt 2007 (paper "Divisible E-Cash Systems Can Be Truly Anonymous").

payment. Therefore, to perform a payment worth of v, one has to use separate coins whose values add up exactly to v. It would be desirable to have some more convenient solution.

This is why the concept of *divisibility* was introduced [18]. It states that a coin can be subdivided into many pieces that can be spent independently and whose values add up to the original value of the coin. Efficient divisible electronic cash schemes enable paying any amount of cash by running the payment protocol $O(\log N)$ times, where N is the *divisibility precision*.

Since its introduction, divisible digital cash was studied by many authors [7, 17, 18, 13].

Another solution is to use compact wallets [4]. Such a wallet enables storing N coins of the same value by storing a couple of numbers of length K for which problems like factorization or discrete logarithm are intractable (they should have a couple hundreds of bytes). Conducting a single payment takes time proportional to $\mathrm{poly}(K) \cdot \mathrm{polylog}(N)$. However, spending multiple coins stored in a wallet takes time linear in the number of coins being spent, which is much worse than the complexity of a divisible scheme (with N as the divisibility precision). For example, paying 100$ using a wallet that stores coins worth 1 cent requires 10000 executions of a payment protocol, while in the same setting our scheme runs the payment protocol just 5 times (the number of 1's in the binary representation of 10000).

Divisible electronic cash schemes of previous authors have their flaws as well. Most (e.g. [17], [7]) don't achieve *unlinkability*, i.e., it is easy to link all payments coming from the same coin or user. Identifying the user remains hard from the cryptographic point of view, but other techniques (like checking the locality of payments, their value etc.) may lead to tracing the user without breaking the cryptosystem the scheme is based on. Nakanishi and Sugiyama [16] proposed a scheme where payments are unlinkable (although not fully, as described in our analysis of unlinkability). Their scheme, however, requires the existence of a trusted third party (TTP) to identify double-spenders. This is a big drawback, because no user is anonymous to the TTP, which contradicts the basic principle of digital cash stating that a user behaving correctly shall never be identified.

Our work can be seen as an attempt to join the advantages of schemes presented in [4] and [16]. Namely, we propose a scheme that supports efficient protocols and coin storage, divisibility, anonymity, unlinkability and protection against double-spending, all without the need for a trusted third party.

The big picture of our scheme is as follows. The withdrawal procedure is similar to the one in [4]. Also almost the same is proving that the commitment presented by the user in a store is one of a valid coin. We adopt the binary tree approach to the coin storage and divisible payments. Node values in our tree are computed from the coin secret in a way similar to [16]. They are used to assure that two payments using the same node send the same values related to the node and thus enable detection of double-spending. These values also allow computing the values for the node's all descendants. Identifying a double-spender is similar to that presented in [4], except in our protocol the double-spending equation depends on the coin secret as well as on the node being spent. It also allows to compute double-spending equations for all descendants of a node, therefore allowing identification of double-spenders violating the divisibility rules.

This paper is organized as follows: in Section 2 we describe the requirements for a digital cash protocol, Section 3 gives an overview of the zero knowledge proofs used. The scheme is presented in Section 4. Section 5 provides the security and efficiency analysis of the proposed scheme. Section 6 concludes the paper.

2 Requirements

We now briefly describe the conditions that we think an unlinkable divisible digital cash scheme should satisfy. The list (based on [16]) is as follows:

Anonymity. The payer who didn't violate the protocol can't be identified from the information sent during payment.

Unlinkability. It's infeasible to link any two payments executed by the same user (even without learning the payer's identity), unless the payments lead to double-spending.

Unforgeability. Neither a coin nor a transcript of a payment of an honest user can be forged.

Exculpability. A user can be identified as a double-spender only if she really is guilty. Additionally, even if a user was caught double-spending, she is only responsible for payments she indeed double-spent (i.e., she cannot be framed into making other payments).

Divisibility. User can perform any payment worth no more than the coin's total value (as long as it is a multiplicity of the minimal value subdivided from the coin).

No double-spending. All double-spending attempts will be detected and the user who tries to spend more than the coin's total value will be identified.

3 Preliminaries

3.1 Pedersen Commitment

In our protocols, we make use of the Pedersen commitment scheme [19]. Let G be a group of prime order p and $(g_1, \ldots, g_n, g_{n+1})$ be its generators. To commit to the values $(v_1, \ldots, v_n) \in \mathbb{Z}_p^n$, the user randomly chooses $r \in \mathbb{Z}_p$ and computes $C = (\prod_{i=1}^{n} g_i^{v_i})\, g_{n+1}^r$ as her commitment.

3.2 Zero knowledge proofs

In this paper we use the notation introduced in [6] to describe zero knowledge proofs for proving statements about discrete logarithms. This notation is as follows: $PK\{(\alpha, \beta, \ldots) : Pred_1 \wedge Pred_2 \wedge \ldots\}$ denotes a zero knowledge proof of knowledge of values $\alpha, \beta, \ldots$ that satisfy predicates $Pred_1, Pred_2, \ldots$, where they appear as exponents. In general, Greek letters denote values that we are proving to know. We now describe few kinds of proofs which are used in our construction. All of these proofs are secure under the Discrete Logarithm Assumption.

3.2.1 Proof of knowledge of representation

Let G be a group and $g_1, \ldots, g_k \in G$. We call these values *bases*. We wish to prove the knowledge of representation of the values $y_1, \ldots, y_n \in G$ to these bases. The corresponding proof is denoted as

$$PK\big\{(\alpha_1, \ldots, \alpha_m) : y_1 = \prod_{j=1}^{l_1} g_{b_{1j}}^{\alpha_{e_{1j}}} \wedge \ldots \wedge y_n = \prod_{j=1}^{l_n} g_{b_{nj}}^{\alpha_{e_{nj}}}\big\},$$

where $l_i \in \{1, \ldots, k\}$ is the number of bases in the representation of y_i, $e_{ij} \in \{1, \ldots, m\}$ are indices to values from $\{\alpha_1, \ldots, \alpha_m\}$ that appear as exponents in the representation of y_i, and $b_{ij} \in \{1, \ldots, k\}$ are indices to values from $\{g_1, \ldots, g_k\}$ that appear as bases in the representation of y_i. Note that the proof of knowledge of discrete logarithm is an instance of this type of proofs.

An example of this kind of proofs is

$$PK\big\{(\alpha_1, \alpha_2, \alpha_3) : y_1 = g_1^{\alpha_1} g_2^{\alpha_2} \wedge y_2 = g_2^{\alpha_2} g_3^{\alpha_3} \wedge y_3 = g_1^{\alpha_3}\big\}.$$

An efficient construction of such proofs is described in [6].

3.2.2 Proof that two values committed in different groups are in $\{0,1\}$ and are equal

The following proof is in part based on the proof that a committed value is in $\{0,1\}$ by Damgård [1]. Our proof can be seen as simultaneously executing two instances (one for each commitment) of a similar proof, but for common challenges (to assure that the commitments hide the same value).

Let G, $\widetilde{G}$ be groups of prime orders p and $\tilde{p}$, respectively (without the loss of generality we can assume that $p \leq \tilde{p}$). The bases for commitments in these groups are $g_1, g_2 \in G$ and $\tilde{g}_1, \tilde{g}_2 \in \widetilde{G}$. This protocol enables proving knowledge of values $a \in \mathbb{Z}_p, \tilde{a} \in \mathbb{Z}_{\tilde{p}}$ and $b \in \{0,1\}$ such that $y = g_1^b g_2^a$ and $\tilde{y} = \tilde{g}_1^b \tilde{g}_2^{\tilde{a}}$. It makes use of the proof

$$PK\{(\alpha, \tilde{\alpha}) : z = g_2^{\alpha} \wedge \tilde{z} = \tilde{g}_2^{\tilde{\alpha}}\},$$

which is as follows: Alice (prover) first chooses random $r_1 \in \mathbb{Z}_p$, $r_2 \in \mathbb{Z}_{\tilde{p}}$ and sends $g_2^{r_1}$, $\tilde{g}_2^{r_2}$ to Bob (verifier). He replies with a challenge $c \in \mathbb{Z}_p$. Then Alice sends $y_1 = r_1 - ca \bmod p$ and $y_2 = r_2 - c\tilde{a} \bmod \tilde{p}$. Bob accepts the proof if $g_2^{r_1} = z^c g_2^{y_1}$ and $\tilde{g}_2^{r_2} = \tilde{z}^c \tilde{g}_2^{y_2}$.

Note that when the claim doesn't hold, Alice can execute the protocol if she knows the challenge c in advance (namely, she chooses y_1 and y_2 and in the first step of communication sends values $z^c g_2^{y_1}$ and $\tilde{z}^c \tilde{g}_2^{y_2}$ to Bob). Also note that

$$PK\{(\alpha, \tilde{\alpha}) : z = g_1 g_2^{\alpha} \wedge \tilde{z} = \tilde{g}_1 \tilde{g}_2^{\tilde{\alpha}}\} \equiv PK\{(\alpha, \tilde{\alpha}) : z/g_1 = g_2^{\alpha} \wedge \tilde{z}/\tilde{g}_1 = \tilde{g}_2^{\tilde{\alpha}}\}.$$

Therefore,

$$PK\{(\alpha, \tilde{\alpha}) : (z = g_1 g_2^{\alpha} \wedge \tilde{z} = \tilde{g}_1 \tilde{g}_2^{\tilde{\alpha}}) \vee (z/g_1 = g_2^{\alpha} \wedge \tilde{z}/\tilde{g}_1 = \tilde{g}_2^{\tilde{\alpha}})\}$$

can be composed from the two above proofs (i.e., $PK\{(\alpha, \tilde{\alpha}) : z = g_2^{\alpha} \wedge \tilde{z} = \tilde{g}_2^{\tilde{\alpha}}\}$ and $PK\{(\alpha, \tilde{\alpha}) : z/g_1 = g_2^{\alpha} \wedge \tilde{z}/\tilde{g}_1 = \tilde{g}_2^{\tilde{\alpha}}\}$) using the disjunction trick [12] in the following way. Without the loss of generality we can assume that the first claim doesn't hold. Alice chooses challenge $c_1 \in \mathbb{Z}_p$, for which she constructs a fake transcript of this proof. She sends the commitment from this transcript and a commitment for the second proof. For Bob's challenge c she gives second proof's response to challenge $c_2 = c - c_1 \bmod p$. Bob accepts the proof if both proofs verify and $c_1 + c_2 = c \bmod p$.

This proof is denoted as

$$PK\{(\alpha, \tilde{\alpha}, \beta) : y = g_1^{\beta} g_2^{\alpha} \wedge \tilde{y} = \tilde{g}_1^{\beta} \tilde{g}_2^{\tilde{\alpha}} \wedge \beta \in \{0,1\}\}.$$

3.2.3 Proof of equality of two values committed in different groups

As before, we have groups G and $\widetilde{G}$ of order p and $\tilde{p}$, respectively, and bases $g_1, g_2 \in G$, $\tilde{g}_1, \tilde{g}_2 \in \widetilde{G}$. Using the following protocol Alice is able to prove the knowledge of values $a, b \in \mathbb{Z}_p$ and $\tilde{a} \in \mathbb{Z}_{\tilde{p}}$, such that $y = g_1^b g_2^a$ and $\tilde{y} = \tilde{g}_1^b \tilde{g}_2^{\tilde{a}}$.

This proof is denoted as

$$PK\{(\alpha, \tilde{\alpha}, \beta) : y = g_1^{\beta} g_2^{\alpha} \wedge \tilde{y} = \tilde{g}_1^{\beta} \tilde{g}_2^{\tilde{\alpha}} \wedge 0 \leq \beta < p\}$$

and can be obtained using the previous proof as a subroutine in the following way.

First, let's note that any value $x \in \mathbb{Z}_p$ can be represented by $(x_0, \ldots, x_k) \in \{0,1\}^{k+1}$, such that $x = x_0 + 2x_1 + 2^2 x_2 + \cdots + 2^{k-1} x_{k-1} + (p - 2^k) x_k$ where $k = \lfloor \lg x \rfloor$ (this representation does not need to be unique). The idea is that Alice first computes the representation $(b_0, \ldots, b_k)$ of b and makes commitments to b_i's in the two groups:

$$C_i = \left(g_1^{2^i}\right)^{b_i} g_2^{r_i}, \quad \widetilde{C}_i = \left(\tilde{g}_1^{2^i}\right)^{b_i} \tilde{g}_2^{\tilde{r}_i} \qquad i = 0, \ldots, k-1$$

$$C_k = \left(g_1^{p-2^k}\right)^{b_k} g_2^{r_k}, \quad \widetilde{C}_k = \left(\tilde{g}_1^{p-2^k}\right)^{b_k} \tilde{g}_2^{\tilde{r}_k}$$

(for random $r_i \in \mathbb{Z}_p$, $\tilde{r}_i \in \mathbb{Z}_{\tilde{p}}$) and proves that these commitments open to the same values b_i:

$$PK\Big\{(\alpha_i, \tilde{\alpha}_i, \beta_i) : C_i = \left(g_1^{2^i}\right)^{\beta_i} g_2^{\alpha_i} \wedge \widetilde{C}_i = \left(\tilde{g}_1^{2^i}\right)^{\beta_i} \tilde{g}_2^{\tilde{\alpha}_i} \wedge \beta_i \in \{0,1\}\Big\} \qquad i = 0, \ldots, k-1$$

$$PK\Big\{(\alpha_k, \tilde{\alpha}_k, \beta_k) : C_k = \left(g_1^{p-2^k}\right)^{\beta_k} g_2^{\alpha_k} \wedge \widetilde{C}_k = \left(\tilde{g}_1^{p-2^k}\right)^{\beta_k} \tilde{g}_2^{\tilde{\alpha}_k} \wedge \beta_k \in \{0,1\}\Big\}$$

Finally, she proves that $(b_0, \ldots, b_k)$ is a representation of b in y:

$$PK\Big\{(\alpha, \alpha', \beta) : \prod_{i=0}^{k} C_i = g_1^{\beta} g_2^{\alpha'} \wedge y = g_1^{\beta} g_2^{\alpha}\Big\}$$

and in $\tilde{y}$:

$$PK\Big\{(\tilde{\alpha}, \tilde{\alpha}', \beta) : \prod_{i=0}^{k} \widetilde{C}_i = \tilde{g}_1^{\beta} \tilde{g}_2^{\tilde{\alpha}'} \wedge \tilde{y} = \tilde{g}_1^{\beta} \tilde{g}_2^{\tilde{\alpha}}\Big\}.$$

3.2.4 Proof that a committed value is a discrete logarithm of another committed value

Let $G, \widetilde{G}$ be groups of prime orders p and $\tilde{p}$, respectively $(p|\tilde{p}-1)$ such that G is a subgroup of $\mathbb{Z}_{\tilde{p}}^*$. Given $y, g_1, g_2, h \in G$ and $\tilde{y}, \tilde{g}, \tilde{h} \in \widetilde{G}$, this protocol enables proving knowledge of values $a, b_1 \in \mathbb{Z}_p, b_2 \in \mathbb{Z}_{\tilde{p}}$ such that $y = g_1^a h^{b_1}$ and $\tilde{y} = \tilde{g}^{g_2^a} \tilde{h}^{b_2}$. Its construction is based on a proof that appeared in [20] and requires K rounds (for a security parameter K) to be executed between Alice (prover) and Bob (verifier), where each round is as follows:

1. Alice selects random $w, r_1 \in \mathbb{Z}_p$ and $r_2 \in \mathbb{Z}_{\tilde{p}}$. She then sends Bob values $t_1 = g_1^w h^{r_1}$ and $t_2 = \tilde{g}^{g_2^w} \tilde{h}^{r_2}$.
2. Bob responds with $c \in \{0,1\}$.
3. if $c = 0$, Alice reveals w, r_1 and r_2 to show that she has generated t_1, t_2 in the right way; Bob checks that $t_1 = g_1^w h^{r_1}$ and $t_2 = \tilde{g}^{g_2^w} \tilde{h}^{r_2}$.
4. if $c = 1$, Alice sends values $A = w - a \bmod p$, $B = r_1 - b_1 \bmod p$, $C = r_2 - b_2 g_2^{w-a} \bmod \tilde{p}$; Bob checks that $t_1 = g_1^A y h^B$ and $t_2 = \tilde{y}^{g_2^A} \tilde{h}^C$.

This proof is denoted as

$$PK\Big\{(\alpha, \beta_1, \beta_2) : y = g_1^{\alpha} h^{\beta_1} \wedge \tilde{y} = \tilde{g}^{g_2^{\alpha}} \tilde{h}^{\beta_2}\Big\}.$$

3.2.5 Proof that a committed value is a discrete logarithm of one of two committed values

This proof is denoted as

$$PK\Big\{(\alpha, \beta_1, \beta_2) : y = g_1^{\alpha} h^{\beta_1} \wedge \left(\tilde{y} = \tilde{g}^{g_2^{\alpha}} \tilde{h}^{\beta_2} \vee \tilde{y} = \tilde{g}^{g_3^{\alpha}} \tilde{h}^{\beta_2}\right)\Big\}$$

and can easily be obtained from the previous proof. To see this, let's note that this proof can equivalently be written as

$$PK\Big\{(\alpha, \beta_1, \beta_2) : \left(y = g_1^{\alpha} h^{\beta_1} \wedge \tilde{y} = \tilde{g}^{g_2^{\alpha}} \tilde{h}^{\beta_2}\right) \vee \left(y = g_1^{\alpha} h^{\beta_1} \wedge \tilde{y} = \tilde{g}^{g_3^{\alpha}} \tilde{h}^{\beta_2}\right)\Big\}.$$

The above proof can be carried out with the use of the disjunction trick [12] as before (namely, fix a challenge c' for one of the proofs, then on challenge c execute one proof for c' and the other for $c'' = c - c'$).

3.3 CL signatures

Let p be a prime, G be a cyclic group of order p, and $\tilde{g}_1, \tilde{g}_2, \tilde{g}_3 \in G$. Let $\tilde{g}_1^s \tilde{g}_2^x \tilde{g}_3^r$ be a Pedersen commitment to (s, x). Just like [4], in our protocols we use the signature scheme introduced by Camenisch and Lysyanskaya [5] and refer to it as the "CL signatures". The main advantage of CL signatures lies in two efficient protocols:

- a protocol for obtaining a blind signature on (s, x) (i.e., the user presents the signer with a Pedersen commitment to (s, x), and the signer signs (s, x) without learning anything about it);
- a protocol for proving the knowledge of a CL signature (i.e., the user presents the verifier with a Pedersen commitment to (s, x) and proves that she knows the signer's CL signature on (s, x) without revealing anything about (s, x) or the signature itself).

CL signatures are secure under the Strong RSA Assumption. We don't discuss how CL signatures actually work as all we need are the protocols described above.

3.4 Binary tree approach

As in the previous divisible digital cash protocols, we utilize the binary tree approach. It states that a coin worth of 2^{l-1} dollars is represented by a binary tree of l levels, where leaves represent unitary amounts (one dollar) of cash and each internal node's value is the sum of the values assigned to its children. Nodes in such a tree are numbered in the standard way, i.e., n_0 is the root node and $n_{i_1 \ldots i_j 0}$ and $n_{i_1 \ldots i_j 1}$ are respectively left and right children of $n_{i_1 \ldots i_j}$. Spending a value 2^{l-j} dollars is realized by revealing some values assigned to a node $n_{i_1 \ldots i_j}$. To prevent double-spending, the following divisibility rule is introduced:

- a user can spend at most one node on each root to leaf path;

We make use of this approach in a slightly modified way. We add an auxiliary level to the tree adding two children to each leaf obtaining $l + 1$ levels instead of l. Monetary values are assigned in the standard way to all internal nodes, i.e., nodes beyond the auxiliary level. Based on the coin secret x, x-values for each node are computed in a way similar to [16]. Figure 1 presents four highest levels of a tree representing 1024$.

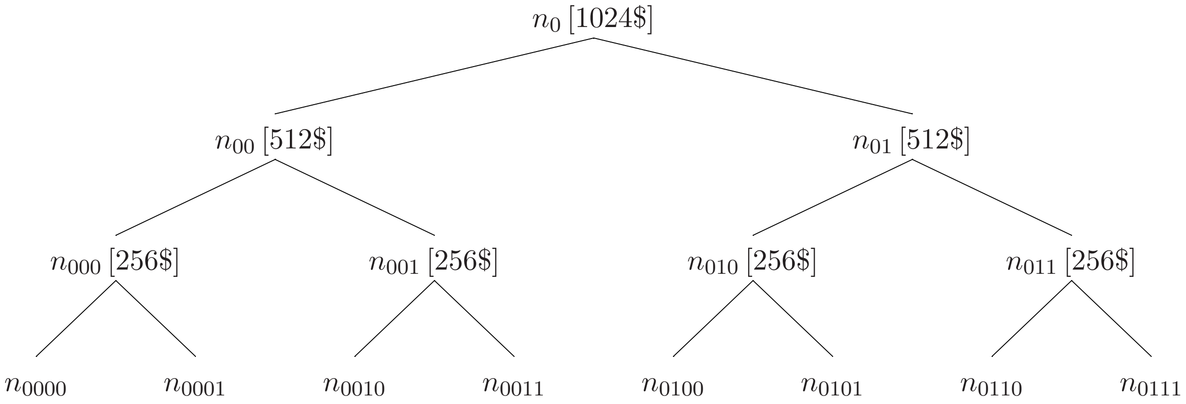

Figure 1: Tree representing a coin worth of 1024$

During the payment with a node $n_{i_1 \ldots i_j}$, x-values for its children are revealed ($n_{i_1 \ldots i_j}$'s x-value is kept secret). The idea is that from a node's x-value all x-values for its descendants can be computed. The size of the numbers in the protocol assures with high probability that all x-values of all nodes in all circulating coins are different. So if two x-values of the leaves appearing in different transactions are the same it is the sign that double-spending occurred.

In our construction in order to identify a double-spender an additional value for a spent node is revealed. It is called the T-value and is based on the T-value used in [4]. It is chosen in a way that enables computing T-values for all descendants of a node whose T-value is revealed. This T-value includes randomness (provided by a cryptographic hash function). So when double-spending takes place, two T-values for a node (for which known child x-values are the same) aren't equal with overwhelming probability and thus allow revealing the identity of a double-spender.

4 The protocol

Setup. This stage is executed by a bank only once for each coin value $v = 2^{l-1}$ dollars.

1. The bank generates a chain of prime numbers $p_1, p_2, \ldots, p_{l+1}$ such that there exist values $\nu_i < 2^k$ and $p_{i+1} = \nu_i p_i + 1 \quad (i = 1, \ldots, l)$, where k is a parameter.
   Furthermore, the bank generates groups $G_{p_1}, \ldots, G_{p_l}$ such that G_{p_i} is a subgroup of order p_i of $\mathbb{Z}^*_{p_{i+1}}$ and $G_{p_i} =< g_{p_i} >$ (G_{p_i} is a cyclic group generated by g_{p_i}).
   Then the bank chooses elements $h_{(1,0)}$, $h_{(1,1)} \in G_{p_1}$, $h_{(2,0)}$, $h_{(2,1)} \in G_{p_2}$, ..., $h_{(l,0)}$, $h_{(l,1)} \in G_{p_l}$ such that discrete logarithms between them and their respective group's generators are unknown.
2. The bank creates an empty database for storing the history of deposited payments. Each payment's entry contains x-values and T-values for all leaves that are descendants of a spent node and an additional value R (a pseudorandom number used in the payment protocol – it will be discussed later).
   The database should also enable efficient searching for x-values (e.g., it could be sorted by x-values).
3. The bank sets up parameters for CL signatures (i.e., $\mathbf{n} = \mathbf{pq}$ and a cyclic group $\mathbf{G}$ of order $\mathbf{n}$).
4. The bank chooses a group G of prime order p ($p > p_{l+1}$), and picks its generators $g, \hat{g} \in G$.

Then the bank makes all generated values public.

Open account. This is executed only once between a user U and a bank.

1. The user randomly chooses her secret key $s \in \mathbb{Z}_p$ and computes her public key $pk_U = \hat{g}^s \in G$.
2. The user sends pair of values (pk_U, ID_U) for identification purposes and keeps s secret.

Withdrawal. This is executed when a user U wishes to withdraw a coin worth of 2^{l-1}. This part is almost the same as in the scheme presented in [4].

1. The user goes to the bank to get a coin. Both the user and the bank contribute randomness to the coin's serial number x. Let $\bar{g}$ and $\bar{h}$ be generators for G_{p_1}. The user randomly selects $x' \in \mathbb{Z}_{p_1}$ and commits to it by computing $A' = \bar{g}^{x'}\bar{h}^{r'}$ for $r' \in \mathbb{Z}_{p_1}$ and sending A' to the bank. The bank responds with random $x'' \in \mathbb{Z}_{p_1}$. The user computes $x = x' + x'' \bmod p_1$. Both the user and the bank independently compute $A = A'\bar{g}^{x''} = \bar{g}^x\bar{h}^{r'}$.
2. The user computes Pedersen commitment to (s, x): $C = \tilde{g}_1^s \tilde{g}_2^x \tilde{g}_3^r$ for $\tilde{g}_1, \tilde{g}_2, \tilde{g}_3 \in G$ and random $r \in \mathbb{Z}_p$ and sends it to the bank. First, she proves that the x used in C is appropriate:

$$PK\{(\sigma, \rho, \rho', \xi) : A = \bar{g}^\xi \bar{h}^{\rho'} \wedge C = \tilde{g}_1^\sigma \tilde{g}_2^\xi \tilde{g}_3^\rho \wedge 0 \leq \xi < p_1\}.$$

Then she proves that s in C is the same as in pk_U:

$$PK\{(\sigma, \rho, \xi) : pk_U = \hat{g}^{\sigma} \wedge C = \tilde{g}_1^{\sigma} \tilde{g}_2^{\xi} \tilde{g}_3^{\rho}\}.$$

3. Using the protocol for CL signatures on C, the user gets the bank's signature on values (s, x).

Coin representation. After executing the withdrawal protocol, a user has a coin, i.e., bank's CL signature on (s, x). Since explicitly revealing x violates at least the unlinkability requirement, spending has to be done in some other fashion.

In our scheme a coin is represented using a binary tree as shown in Figure 1. With each node of that tree we associate an x-value, which depends only on x and the node's location in the tree. By $x_{i_1 \dots i_j}$ we denote the x-value associated with node $n_{i_1 \dots i_j}$. The computation of x-values goes as follows:

$$\begin{aligned} x_0 &= x \\ x_{i_1 \dots i_j i_{j+1}} &= h_{(j, i_{j+1})}^{x_{i_1 \dots i_j}} \qquad j = 1, \dots, l \end{aligned}$$

Figure 2 shows an example of a tree with x-values computed this way.

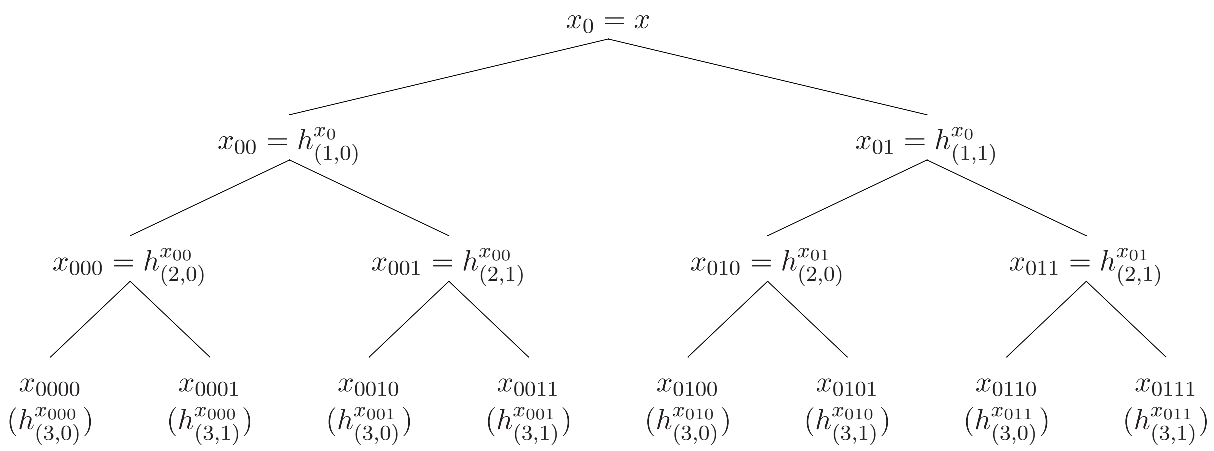

Figure 2: x-values for the tree

Payment. A user wishes to spend the node $n_{i_1 i_2 \dots i_j}$ of the coin whose secrets are (s, x) to a merchant. Doing this she pays 2^{l-j} dollars. The node's address $(i_1 \dots i_j)$ isn't revealed. The user shows that values sent during that payment are generated from the coin secret x in the correct way applying zero knowledge proofs.

Note that we only describe how to spend a certain node, not how to choose which nodes to spend to realize a payment.

1. The user generates a Pedersen commitment to (s, x): $C = \tilde{g}_1^{s} \tilde{g}_2^{x} \tilde{g}_3^{r}$ and, using protocol for CL signatures, proves that she knows bank's signature on (s, x).
2. Both the user and the merchant compute $R \in \mathbb{Z}_p$ by putting data like merchant's identity, exact time of transaction etc. as an input to a cryptographic hash function.
3. The user sends values $T_{i_1 \dots i_j} = (g^{x_{i_1} + \dots + x_{i_1 \dots i_j}})^R \hat{g}^s$, $x_{i_1 \dots i_j 0}$ and $x_{i_1 \dots i_j 1}$ to the merchant.
4. The user proves that the values $x_{i_1 \dots i_j 0}$ and $x_{i_1 \dots i_j 1}$ are generated from the x that appeared in C and according to the protocol. To do this, she chooses bases for Pedersen

commitments in $G_{p_1}, \ldots, G_{p_j}$: $g_1, h_1 \in G_{p_1}, \ldots, g_j, h_j \in G_{p_j}$, randomly selects values $r_1 \in \mathbb{Z}_{p_1}$, $r_2 \in \mathbb{Z}_{p_2}, \ldots, r_j \in \mathbb{Z}_{p_j}$ and computes Pedersen commitments to the x-values on the path from the root to the node being spent: $C_1 = g_1^{x_{i_1}} h_1^{r_1}$, $C_2 = g_2^{x_{i_1 i_2}} h_2^{r_2}$, $\ldots$, $C_j = g_j^{x_{i_1 \ldots i_j}} h_j^{r_j}$.The user then sends values $(g_1, h_1, \ldots, g_j, h_j, C_1, \ldots, C_j)$ to the merchant. Now the user is ready to execute following proofs of knowledge.
First, she proves that the root's x-value is the same as x in the opening of C:

$$PK\{(\sigma, \rho, \rho_1, \xi) : C = \tilde{g}_1^{\sigma} \tilde{g}_2^{\xi} \tilde{g}_3^{\rho} \wedge C_1 = g_1^{\xi} g_2^{\rho_1} \wedge 0 \leq \xi < p_1\}.$$

Then she executes the proofs for x-values on the path (for $i = 1, \ldots, j-1$):

$$PK\big\{(\xi_{i-1}, \rho_i, \rho_{i+1}) : C_i = g_i^{\xi_{i-1}} h_i^{\rho_i} \wedge \big(C_{i+1} = g_{i+1}^{h_{(i+1,0)}^{\xi_{i-1}}} h_{i+1}^{\rho_{i+1}} \vee C_{i+1} = g_{i+1}^{h_{i+1,1}^{\xi_{i-1}}} h_{i+1}^{\rho_{i+1}}\big)\big\}$$

Finally, the user proves correctness of $x_{i_1 \ldots i_j 0}$ and $x_{i_1 \ldots i_j 1}$:

$$PK\big\{(\xi_{j-1}, \rho_j) : C_j = g_j^{\xi_{j-1}} h_j^{\rho_j} \wedge x_{i_1 \ldots i_j 0} = h_{(j,0)}^{\xi_{j-1}}\big\}$$
$$PK\big\{(\xi_{j-1}, \rho_j) : C_j = g_j^{\xi_{j-1}} h_j^{\rho_j} \wedge x_{i_1 \ldots i_j 1} = h_{(j,1)}^{\xi_{j-1}}\big\}.$$

5. The user proves that her secret key s from C is used also in $T_{i_1 \ldots i_j}$:

$$PK\{(\alpha, \sigma, \xi, \rho) : C = \tilde{g}_1^{\sigma} \tilde{g}_2^{\xi} \tilde{g}_3^{\rho} \wedge T_{i_1 \ldots i_j} = \big(g^R\big)^{\alpha} \hat{g}^{\sigma}\}.$$

6. The user proves that the power of g^R in $T_{i_1 \ldots i_j}$ is of the right form. To do this, she computes commitments $\widetilde{C}_1 = \tilde{g}^{x_{i_1}} \tilde{h}^{\tilde{r}_1}$, $\widetilde{C}_2 = \tilde{g}^{x_{i_1 i_2}} \tilde{h}^{\tilde{r}_2}$, $\ldots$, $\widetilde{C}_j = \tilde{g}^{x_{i_1 \ldots i_j}} \tilde{h}^{\tilde{r}_j}$ for $\tilde{g}, \tilde{h} \in G$ and random $\tilde{r}_1, \ldots, \tilde{r}_j \in \mathbb{Z}_p$. Then she executes the proof:

$$PK\big\{(\xi, \rho, \sigma) : \prod_{i=1}^{j} \widetilde{C}_i = \tilde{g}^{\xi} \tilde{h}^{\rho} \wedge T_{i_1 i_2 \ldots i_j} = (g^R)^{\xi} \hat{g}^{\sigma}\big\}$$

and proves that $\widetilde{C}_1, \ldots, \widetilde{C}_j$ are correct (hide the same x-values as C_1, $\ldots$, C_j):

$$PK\{(\xi_0, \rho_1, \tilde{\rho}_1) : C_1 = g_1^{\xi_0} h_1^{\rho_1} \wedge \widetilde{C}_1 = \tilde{g}^{\xi_0} \tilde{h}^{\tilde{\rho}_1} \wedge 0 \leq \xi_0 < p_1\}$$
$$\vdots$$
$$PK\{(\xi_{j-1}, \rho_j, \tilde{\rho}_j) : C_j = g_j^{\xi_{j-1}} h_j^{\rho_j} \wedge \widetilde{C}_j = \tilde{g}^{\xi_{j-1}} \tilde{h}^{\tilde{\rho}_j} \wedge 0 \leq \xi_{j-1} < p_j\}.$$

Note: The Fiat-Shamir heuristic [14] turns interactive proofs of knowledge into non-interactive ones. It works the following way: the prover determines the challenge by applying a cryptographic hash function $\mathcal{H}$ to the commitments that appear in the proof and some additional values (R in our case) and then responds to this challenge.

We use this heuristic on all proofs that appear in our protocol to assure the unforgeability of a payment's transcript.

Deposit. It is executed between a merchant (who wants to exchange digital cash received during payment into true money) and a bank.

1. The merchant gives the bank a transcript of a payment. The bank checks its validity and, if it is correct, goes to the following step.
2. Let the spent node be $n_{i_1 i_2 \ldots i_j}$. From the values $x_{i_1 i_2 \ldots i_j 0}$ and $x_{i_1 i_2 \ldots i_j 1}$ the bank is able to compute x-values and T-values for all leaves whose ancestor is $n_{i_1 i_2 \ldots i_j}$ (note

that

$$\begin{aligned} T_{i_1 i_2 \ldots i_j b} &= \left(g^{x_{i_1}+\cdots+x_{i_1\ldots i_j}+x_{i_1\ldots i_j b}}\right)^R pk_U \\ &= \left(\left(g^{x_{i_1}+\cdots+x_{i_1\ldots i_j}}\right)^R pk_U\right)\left(g^{x_{i_1\ldots i_j b}}\right)^R \\ &= T_{i_1\ldots i_j}\left(g^R\right)^{x_{i_1\ldots i_j b}} \end{aligned}$$

for $b \in \{0,1\}$). If any of those x-values already appeared in bank's X database, trigger the identification of double-spenders algorithm, otherwise add x-values, T-values and R to the database.

Identification of double-spenders. Bank needs to identify a double-spender if in its database there are two identical x's.

We argue that an accidental equality of these values for different leaves or different coins is almost impossible. We can treat x's as chosen at random, so the birthday paradox' analysis applies. For example, if the primes used are K bits long and bank issues A coins, each with a divisibility precision of N, then the probability of a collision is small as long as $A \cdot N \ll 2^{K/2}$.

In such case the corresponding T_1, T_2 on level l and R_1, R_2 are also in the bank database. We have $T_1 = \left(g^{x_{i_1}+\cdots+x_{i_1\ldots i_l}}\right)^{R_1} pk_U$, $T_2 = \left(g^{x_{i_1}+\cdots+x_{i_1\ldots i_l}}\right)^{R_2} pk_U$. With huge probability $R_1 \neq R_2$. Bank can recover pk_U by computing

$$\left(T_1^{R_2}/T_2^{R_1}\right)^{(R_2-R_1)^{-1}} = pk_U^{(R_2-R_1)(R_2-R_1)^{-1}} = pk_U$$

thus the identity of a double-spender is revealed.

5 Analysis

5.1 Security and other properties

5.1.1 Cryptographic assumptions

First we state the cryptographic assumptions on which the security of our system depends.

Strong RSA Assumption [4]. Given an RSA modulus n and a random element $g \in \mathbb{Z}_n^*$, it is hard to compute $h \in \mathbb{Z}_n^*$ and an integer $e > 1$ such that $h^e \equiv g \bmod n$. The modulus n is of a special form pq, where $p = 2p' + 1$ and $q = 2q' + 1$ are safe primes.

Discrete Logarithm Assumption. Given elements $g, h \in G$ it is hard to compute x such that $h = g^x$.

Hash functions. In order to use the Fiat-Shamir heuristic we assume the existence of cryptographic hash functions.

Our assumption. Let $p_1, p_2, \ldots, p_l, p_{l+1}$ be a chain of primes such that there exist values ν_i and $p_{i+1} = \nu_i p_i + 1$ for $i = 1, \ldots, l$, let G be a group of a prime order p ($p > p_{l+1}$) generated by g and let G_{p_i} be a subgroup of $\mathbb{Z}_{p_{i+1}}$ of order p_i for $i = 1, \ldots, l$. Furthermore, let $h_{(1,0)}, h_{(1,1)} \in G_{p_1}$, $h_{(2,0)}, h_{(2,1)} \in G_{p_2}$, …, $h_{(l,0)}, h_{(l,1)} \in G_{p_l}$ (our scheme's h-values).

Our assumption can be formulated as follows. Let $x_1, x_2, \ldots, x_j$ be a chain of x-values that come from some x_1 (i.e., $x_{i+1} = h_{(i,b_i)}^{x_i}$ for $b_i \in \{0,1\}$). We are given three values: $h_{(j,0)}^{x_j}, h_{(j,1)}^{x_j}$

and $t = g^{x_1+\cdots+x_j}$, and three other values: $h_{(k,0)}^{x'_k}, h_{(k,1)}^{x'_k}$ and t' where $x'_1, x'_2, \ldots, x'_k$ is another chain of x-values.

The assumption states that there is no probabilistic polynomial-time algorithm that distinguishes triples for which $x'_1 = x_1$ and $t' = g^{x'_1+\cdots+x'_k}$ from triples for which x'_1 and t' are random in their groups unless $x'_1 = x_1$ and one of x-chains contains another.

Here 'distinguishes' means that the algorithm returns 0 or 1 and 1 is returned substantially more often (in the sense of probability) if the equality holds.

Discussion. Our assumption can be seen as a generalization of the assumption that testing the equality of discrete logarithms is infeasible.

The Nakanishi-Sugiyama protocol [16] depends on a variation of that assumption. In terms of our scheme it could be formulated as follows: Let us have two x-chains: $x_1, x_2, \ldots, x_j$ and $x'_1, x'_2, \ldots, x'_k$, none of which is contained in the other. Their assumption states that knowing only $h_{(j,0)}^{x_j}$ and $h_{(k,0)}^{x'_k}$, it is infeasible to decide whether $x_1 = x'_1$.

This could be generalized by assuming that the additional knowledge of $h_{(j,1)}^{x_j}$ and $h_{(k,1)}^{x'_k}$ does not help.

It could be generalized even further by assuming that infeasibility holds when we add two more values to the input. They are $s = g^{x_j}$ and $s' = g^{x'_k}$ (when $x'_1 = x_1$) or random $s' = g^r$ (when $x'_1 \neq x_1$).

In our assumption we replaced s and s' with two other values. They are $t = s \cdot g^{x_1+\cdots+x_{j-1}}$ and $t' = s' \cdot g^{x'_1+\cdots+x'_{k-1}}$ (or $t' = g^r$). Since we believe there is no special relation between $x_1 + \cdots + x_{j-1}$ and x_j, replacing s by t should not affect feasibility of our decision. The same can be said about s' and t'.

5.1.2 Meeting the requirements

We now show that our scheme meets the requirements stated in section 2 under the aforementioned assumptions.

Anonymity. Note that it is enough to prove unlinkability and anonymity will follow (obviously, if one could identify owners of the payments, he would be able to tell if these payments come from the same user or not). Therefore we shall focus on the unlinkability requirement.

Unlinkability. Recall we mentioned that unlinkability in [16] is not full. The reason is that the addresses of spent nodes are disclosed (and thus if two payments' addresses are in the ancestor-descendant relation and the identification of double-spenders procedure fails, then we know that these payments came from different users), as in our scheme no information is revealed on the location of the node being spent other than the tree level the node is on.

We shall now present the definition of unlinkability. For us, it means that no probabilistic polynomial-time adversary $\mathcal{A}$ representing a coalition of the bank and merchants (behaving accordingly to our scheme's description) has non-negligible advantage in distinguishing two payments executed by the same user from two payments executed by different users.

We will first discuss which values disclosed during the payment with a node $n_{i_1 \ldots i_j}$ are possibly useful for $\mathcal{A}$. Clearly, $x_{i_1 \ldots i_j 0}$, $x_{i_1 \ldots i_j 1}$, R and $T_{i_1 \ldots i_j}$ might be useful. However, it proves that these are the only potentially useful values. The reason is that any other disclosed value is either a Pedersen commitment (which does not reveal anything about its secret) or a part of a transcript of a zero-knowledge proof. We can generate indistinguishable data generating random elements of respective groups as Pedersen commitments and zero-knowledge proof transcripts using a simulator. In the random oracle model these data are equivalent to those obtained from a real transaction.

Therefore any algorithm that would break our scheme's unlinkability (decide whether two payments come from the same user) should work given just these four values for each payment. Due to our cryptographic assumption there are no such algorithms.

Unforgeability. There are three questions to consider when discussing unforgeability:

- can the coin be forged?
- can the payment be realized without the knowledge of a coin?
- can the transcript of a payment be forged?

The answers to all of these questions are "no". Coin's unforgeability follows from the unforgeability of the CL signatures. The payment cannot be realized without a coin because the proof of knowledge of a CL signature (that is a part of the payment protocol) cannot be forged for verifier's random challenge, and note that in our case the challenge is provided by a cryptographic hash function whose output is close to random. The transcript of a payment cannot be forged by a similar argument, since it is infeasible to forge the proof of the knowledge of a user's secret key (that also appears as a part of the payment protocol) for verifier's random challenge.

Exculpability. As shown above, forging the transcript of a payment requires the knowledge of a user's secret key. The protocol for identifying the double-spender only computes her public key, thus providing no useful information on her secret key. Computing the secret key from the values available to the adversary is in contradiction to the Discrete Logarithm Assumption and therefore is infeasible.

Divisibility. As in previous divisible cash schemes, divisibility is accomplished by allowing the user to pay any node of a coin's representation to the merchant.

No double-spending. Our scheme enforces the user to provide the merchant with correct x-values and T-values. If double-spending occurs, then some leaf's x-values for the two payments are the same and double-spending is detected.

As stated before, detected double-spending leads to identifying the double-spender if only the values R for the two payments aren't equal. But this happens with a huge probability because R is an output of the cryptographic hash function depending on the the transaction details among many factors.

5.2 Complexity

The setup stage is executed only once, so even if it wasn't very efficient, it wouldn't hurt very much. However, it takes time proportional to $\mathrm{poly}(K) \cdot \mathrm{polylog}(N)$. The complexity of open account and withdrawal procedures is asymptotically the same as in [4]. To pay an arbitrary amount of cash, one needs to run the payment protocol $O(\log N)$ times. The complexity of spending any node is proportional to $\mathrm{poly}(K) \cdot \mathrm{polylog}(N)$ and is comparable to [16]. The complexity of deposit (without double-spending detection) is the same as that of payment.

The bank's database has to be large enough to store all x-values for all coins ever issued by the bank (i.e., if bank plans on issuing at most A coins with the divisibility precision N, it should have size $O(AN)$). Since the database is sorted, searching for a value and inserting a new value can be done quickly (in time logarithmic in the size of the database). Thus the complexity of double-spending detection is the same as in [4, 16].

Additionally during the deposit it would be enough for the bank just to check the correctness of the data presented by the merchant *on-line*, and then check for double-spending *off-line*.

6 Conclusions and problems

We have presented a divisible digital cash scheme, where unlinkability is achieved without the need for a trusted third party. Our scheme preserves good complexity of previous divisible schemes. It is possible that the complexity of our scheme could be further improved if we applied more efficient zero knowledge protocols than those from subsections 3.2.3 and 3.2.4. The security of our scheme is also based on a strong ad-hoc assumption rather than a well-studied one – we do not know if our assumption reduces to any of well-studied problems.

References

[1] Fabrice Boudot, Berry Schoenmakers, Jacques Traoré: A fair and efficient solution to the socialist millionaires' problem, *Discrete Applied Mathematics* 111, pp. 23-36, Elsevier (2001).

[2] Stefan Brands: Untraceable off-line cash in wallets with observers, *Advances in Cryptology – CRYPTO '93*, LNCS 773, pp. 302-318, Springer Verlag (1994).

[3] Ernest Brickell, Peter Gemmell, David Kravitz: Trustee-based tracing extensions to anonymous cash and the making of anonymous change, *SODA '95*, pp. 457-466, ACM (1995).

[4] Jan Camenisch, Susan Hohenberger, Anna Lysyanskaya: Compact e-cash, *Advances in Cryptology – EUROCRYPT '05*, LNCS 2494, pp. 302-321, Springer Verlag (2005).

[5] Jan Camenisch, Anna Lysyanskaya: A signature scheme with efficient protocols, *Security in Communication Networks '02*, LNCS 2576, pp. 268-289, Springer Verlag (2002).

[6] Jan Camenisch, Markus Stadler: Efficient group signature schemes for large groups, *Advances in Cryptology – CRYPTO '97*, LNCS 1296, pp. 410-424, Springer Verlag (1997).

[7] Agnes Chan, Yair Frankel, Yiannis Tsiounis: Easy come – easy go divisible cash, *Advances in Cryptology – EUROCRYPT '98*, LNCS 1403, pp. 561-575, Springer Verlag (1998).

[8] David Chaum: Blind signatures for untraceable payments. *Advances in Cryptology – CRYPTO '82*, pp. 199-203, Plenum Press (1982).

[9] David Chaum: Security without identification: transaction systems to make big brother obsolete, *Communications of the ACM* 28, 10, pp. 1030-1044 (1985).

[10] David Chaum, Amos Fiat, Moni Naor: Untraceable electronic cash, *Advances in Cryptology – CRYPTO '88*, pp. 319-327, Springer Verlag (1988).

[11] David Chaum, Torben Pedersen: Transferred cash grows in size, *Advances in Cryptology – EUROCRYPT '92*, LNCS 658, pp. 390-407, Springer Verlag (1992).

[12] Ronald Cramer, Ivan Damgård, Berry Schoenmakers: Proofs of partial knowledge and simplified design of witness hiding protocols, *Advances in Cryptology – CRYPTO '94*, LNCS 839, pp. 174-187, Springer Verlag (1994).

[13] Tony Eng, Tatsuaki Okamoto: Single-term divisible electronic coins, *Advances in Cryptology – EUROCRYPT '94*, LNCS 950, pp. 306-313, Springer Verlag (1994).

[14] Amos Fiat, Adi Shamir: How to prove yourself: practical solutions to identification and signature problems, *Advances in Cryptology – CRYPTO '86*, LNCS 263, pp. 186-194 (1986).

[15] Toru Nakanishi, Mitsuaki Shiota, Yuji Sugiyama: An efficient on-line electronic cash with unlinkable exact payments, *7th Information Security Conference – ISC'04*, LNCS 3225, pp. 367-378, Springer Verlag (2004).

[16] Toru Nakanishi, Yuji Sugiyama: Unlinkable divisible electronic cash, *3rd International Workshop on Information Security*, LNCS 1975, pp. 121-134, Springer Verlag (2000).

[17] Tatsuaki Okamoto: An efficient divisible electronic cash scheme, *Advances in Cryptology – CRYPTO '95*, LNCS 963, pp. 438-451, Springer Verlag (1995).

[18] Tatsuaki Okamoto, Kazuo Ohta: Universal electronic cash, *11th Annual International Cryptology Conference on Advances in Cryptology*, LNCS 576, pp. 324-337, Springer Verlag (1991).

[19] Torben Pedersen: Non-interactive and information-theoretic secure verifiable secret sharing, *Advances in Cryptology – CRYPTO '91*, LNCS 576, pp. 129-140, Springer Verlag (1991).

[20] Markus Stadler: Publicly verifiable secret sharing, *Advances in Cryptology – EUROCRYPT '96*, LNCS 1070, Springer Verlag (1996).

A Node spending strategy

So far we have only presented the way of spending a single node. However, in practice one payment has to be realized by spending few nodes, whose values add up to the value of a payment. It is important to choose such a set of nodes for a payment, that using them does not force next payments to use an enormous number of nodes. We shall present a solution to that problem.

The first strategy requires us to assure that there is at most one *open* node on each level of the tree (a node is *open* when none of its children has been spent and its parent is not available to be spent or the node itself is root). When we wish to spend a node on level j, there are two possibilities:

- there is an *open* node on this level – then we use this node;
- there are no *open* nodes on this level – then we choose v – the lowest *open* node above the jth level and spend w – any node on jth level that is a descendant of that node. After realizing this, w's sibling becomes *open*, as well as every sibling of any node on the path from v to w. Additionally, v becomes unavailable. The *open node* property is maintained, since there were no *open* nodes on the levels from j to v's level before.

B Unlinkability

Recall that our goal is to show that no probabilistic polynomial-time adversary $\mathcal{A}$ has non-negligible advantage in distinguishing payments realized by the same user from payments realized by two different users under the aforementioned assumptions.

We already discussed why only $x_{i_1\ldots i_j 0}$, $x_{i_1\ldots i_j 1}$, R and $T_{i_1\ldots i_j}$ may be useful for $\mathcal{A}$.

We are now ready to get to the point. Assume that there exists a randomized polynomial-time algorithm $\mathcal{A}$ that, given two sets of values: $(x_{i_1\ldots i_j 0},\ x_{i_1\ldots i_j 1},\ T_{i_1\ldots i_j},\ R)$ and $(x'_{i'_1\ldots i'_k 0},\ x'_{i'_1\ldots i'_k 1},\ T'_{i'_1\ldots i'_k},\ R')$, has non-negligible advantage in deciding whether these payments come from the same user.

Now let $(h^{x_j}_{(j,0)}, h^{x_j}_{(j,1)}, t = g^{x_1+\cdots+x_j})$ and $(h^{x'_k}_{(k,0)}, h^{x'_k}_{(k,1)}, t')$ be an instance of our assumption. We will show how to use $\mathcal{A}$ to solve it (determine if t' is of the correct form).

First, we randomly choose the public key $pk_U = \hat{g}^u$ of a user whose payments we will simulate. Then we generate the first payment. We set it as $(h^{x_j}_{(j,0)}, h^{x_j}_{(j,1)}, T = t^R\hat{g}^u, R)$, for random R. Computing the second payment is similar: we randomly choose R' and get the second payment: $(h^{x'_k}_{(k,0)}, h^{x'_k}_{(k,1)}, T' = t'^{R'}\hat{g}^u, R')$. We give generated values to $\mathcal{A}$ and return its response.

Why does this work? If $t' = g^{x'_1+\cdots+x'_k}$, then $T' = t'^R\hat{g}^u = g^{R(x'_1+\cdots+x'_k)}\hat{g}^u$. Otherwise, $t' = g^{x'_1+\cdots+x'_k}\hat{g}^v$ for some non-zero v and $T' = t'^R\hat{g}^u = g^{R(x'_1+\cdots+x'_k)}\hat{g}^{Rv+u}$. Since $\hat{g}$ has prime order p, then $\hat{g}^{Rv} \neq 1$ and therefore $\hat{g}^{u'} = \hat{g}^{Rv+u} \neq \hat{g}^{u}$. Value $T = t^R \hat{g}^u$ is always of the form $g^{R(x_1+\cdots+x_j)}\hat{g}^u$, so the $\mathcal{A}$'s response is equivalent to deciding whether $(h_{(j,0)}^{x_j}, h_{(j,1)}^{x_j}, t = g^{x_1+\cdots+x_j})$ and $(h_{(k,0)}^{x'_k}, h_{(k,1)}^{x'_k}, t')$ are of the desirable form $(x_1 = x'_1, t' = g^{x'_1+\cdots+x'_k})$ which solves our cryptographic assumption.