

# Amortized E-Cash

Moses Liskov<sup>1</sup> and Silvio Micali<sup>2</sup>

<sup>1</sup> mliskov@theory.lcs.mit.edu  
<sup>2</sup> silvio@lcs.mit.edu

**Abstract.** We present an e-cash scheme which provides a trade-off between anonymity and efficiency, by amortizing the cost of zero-knowledge and signature computation in the cash generation phase.

Our work solves an open problem of Okamoto in divisible e-cash. Namely, we achieve results similar to those of Okamoto, but (1) based on traditional complexity assumptions (rather than ad hoc ones), and (2) within a much crisper definitional framework that highlights the anonymity properties, and (3) in a simple fashion.

## 1 Introduction

Ever since the work of Chaum [5], there has been an interest in e-cash systems that are not only unforgeable and anonymous but also *off-line*. That is, the merchant can recognize the value of the e-cash without having to check with the bank (unlike, for example, in a credit-card transaction). While very desirable, however, off-line e-cash comes at a price: it cannot totally prevent *double spending*; that is, the ability of a malicious user to spend an “e-coin” more than once. Indeed, an e-coin is a self-verifiable string of bits that is easily reproducible, and therefore could be spent multiple times given the absence of any checking with a central database. Thus, the typical defense envisaged against double spending in off-line e-cash is less than ideal: if double spending occurs, then anonymity is removed and the malicious customer’s identity is revealed.<sup>1</sup>

Most e-cash schemes use zero knowledge computation during e-coin generation to guarantee these complex properties and security requirements.

ZERO-KNOWLEDGE AND SIGNATURES IN E-CASH. To exemplify a common use of zero-knowledge and signatures in off-line e-cash, let us use the proposal of [19]. In their work, an e-coin is authenticated by a blind signature from the bank, so that the bank is unaware of the actual coin it is issuing. As a protection against double spending, the e-coin has the customer’s identity secretly embedded, in a way that enables this identity to be revealed if double spending occurs. Of course, before the bank signs such an e-coin, it must be assured that this secret embedding has been properly done. It is here that zero knowledge plays its role: it enables the bank to check the correctness of the coin structure without learning its specifics, which otherwise would violate the anonymity requirement.

<sup>1</sup> Of course, the possibility of revealing a malicious customer’s identity necessitates the assurance that an honest customer cannot be framed for double spending.

## 1.1 Amortized E-Cash

Signatures and zero-knowledge protocols are the most expensive steps of e-coin generation, and thus make it impractical for coins to have small denominations: such coins would not be worth the computation it takes to generate them. In this paper, therefore, we put forward a method to amortize these expensive steps across *many* coins.

Notice that having the bank validate a list of coins with a single signature, after verifying with zero-knowledge computation that each coin in the list is well-formed, does not provide sufficient amortization: the zero-knowledge computation is simply  $n$  times as large as it was before. For amortization to be at all useful, the efficiency of generating  $n$  new coins should be significantly less than  $n$  times that required for generating a single old coin. Ideally, we should be able to generate  $n$  new coins with essentially the same effort required for a single coin.

Let us give an overview of how our solution achieves amortization and double spending protection.

**AMORTIZATION.** Our solution is based on the notion of a *wallet*. A wallet is a collection of  $n$  coins (strings), each composed of two *logically indivisible* subcoins (substrings). The first subcoin of each coin is *common* to the wallet: it specifies the wallet name and the number of coins in it, secretly embeds the customer's ID, and contains a compact description of all allowable second subcoins. The second subcoin of each coin is *individual*: it contains information essential for spending that coin.

The common subcoin contains all the information the bank verifies by means of a zero-knowledge protocol. Therefore, a single zero-knowledge computation is amortized across many coins. Moreover, the common subcoin is the only data that the bank needs to sign. Therefore, a single signature computation is amortized across many coins.

We shall show that double spending any coin in the wallet causes the customer's identity, secretly embedded in the common subcoin, to be revealed. Moreover, we shall also prove that the common subcoin specifies all the individual subcoins that can be used with it, so that it is impossible for a malicious user to generate unpaid-for coins by using the already bank-signed common subcoin, and then manufacturing additional individual subcoins.

**DOUBLE SPENDING.** Our scheme includes choosing a secret key  $SK$  that is *both* the encryption key of a symmetric-key encryption scheme  $(E, D)$  and the secret signing key of a digital signature scheme with public key  $PK$ .<sup>2</sup>

The way in which we use such a step to protect against double spending is best explained in the simple case in which a wallet contains a single coin.

---

<sup>2</sup> Such "secret key overloading" is potentially dangerous: many security flaws are known to arise when this is done. Thus, we will have to argue that these flaws do not arise in our scheme. We discuss this issue in section 4.4, under "wallet anonymity."

In this case, a customer  $C$  randomly generates a signature key pair  $(PK, SK)$ . Then, the common subcoin of the wallet contains  $PK$  and the customer's identity encrypted with  $SK$ :  $E_{SK}(ID_C)$ . Spending the coin consists of providing a signature (on a challenge) relative to the signature public key  $PK$ . The signature scheme used will be *one-time*, meaning that signing a single message is secure, while signing any two different messages is guaranteed to reveal the secret signing key  $SK$ . Because the secret signing key is also the encryption key (and thus the decryption key), double spending (i.e. signing two different challenges) reveals the customer's identity.

This approach becomes more complex when there are more coins in the same wallet. Having a single, one-time signature public key  $PK$  for all coins in the wallet does not work: legitimately spending two different coins in the wallet would correspond to signing two different challenges relative to the same one-time  $PK$ , thus revealing  $SK$  and therefore, the customer's identity. On the other hand, if each key in the wallet were to have its own key pair  $(PK_i, SK_i)$ , then the common coin would have to include  $E_{SK_i}(ID_C)$  for every  $i$ . Regardless that the common subcoin is growing rapidly in size, this does not provide adequate amortization because each  $E_{SK_i}(ID_C)$  must be verified as correct through a zero-knowledge protocol.

As we shall see, our solution uses  $n$  ephemeral signing keys (one per coin in the wallet) so that any two signatures computed using the same *ephemeral* signing key reveal a master signing key  $SK$ , which is the same as the encryption key used to embed the customer's identity.

**ANONYMITY.** Our scheme satisfies a new type of anonymity requirement: *wallet anonymity*. Informally,

- Coins (from *any* wallet) cannot be linked to a user's identity.
- Coins from *different* wallets (of course, if spent in different transactions) cannot be linked to one another.

However, coins from the *same* wallet can be linked to one another, since they have the same common subcoin. This linkage could be utilized to infer something about the customer from his coin transactions. For example, two merchants, one selling tobacco products and the other gambling products, could compare notes, discover that coins have been spent with each of them that possess the same common subcoin, and thus deduce that they share an anonymous customer (who smokes *and* gambles). In the extreme, imagine that all the coins a customer spends are from the same wallet. Then, if all the merchants he visits compare notes, they might be able to pinpoint the customer not by mathematically breaking the scheme, but with the sheer amount of information about his spending habits.<sup>3</sup>

---

<sup>3</sup> It is worth noting that in practice our scheme will not be that vulnerable to this kind of linking. Small wallets, the certainty of there being some honest merchants, and the low value of most associates make this kind of linking not a great concern.

Though less than perfect, wallet anonymity is useful. Zero-knowledge computation is the true bottleneck of e-cash generation, and our scheme allows e-coins to be generated efficiently, while still providing a good deal of anonymity. Furthermore, our anonymity level can be increased by either (1) having wallets consist of few coins, or (2) having a customer devote different wallets to different type of purchases.

For example, Alice may have one wallet to purchase a given magazine, and a different wallet to purchase electronics. Then, the fact that the same customer keeps on buying the same (or similar) magazines is not much information. Furthermore, because the two wallets are distinct, the producers of the electronics cannot discover that they have a client who reads those magazines, and thus have no incentive to increase their advertising in those magazines.

## 1.2 Amortized E-Cash vs. Divisible E-Cash and Multi-Spendable Coins

The concept of amortized e-cash has been studied previously under the name of “divisible e-cash” ([26], [27], [8], [10], [28]) and “multi-spendable coins” [1]. Divisible e-cash is based on the idea of having a coin which can be divided, at spending, into smaller coins. What we call a wallet is what was previously called a (divisible or multi-spendable) coin, and what we call a coin is merely the atomic part of a coin. However,

1. It is fair to say that most of the research in this area has privileged the algorithmic aspects at the expense of defining what this different concept should mean. No formal treatment of the subtle anonymity properties satisfied by a divisible e-cash scheme has been presented. After being pointed to some “linkability problems,” the reader was largely left alone in figuring out precisely what they were. Indeed, in his 95 paper [26], Okamoto recognizes that the security requirements discussed in his paper are quite ad hoc, and poses as a remaining problem to  
*“Find requirements which are formally shown to be sufficient for the security of electronic cash schemes.”*
2. Prior e-cash schemes such as that of Okamoto [26], were based on *ad hoc* complexity assumptions, that leave in doubt the security of these fast schemes. Indeed, Okamoto poses as an open problem in his paper to  
*“Prove the security under more primitive assumptions such as the hardness of factoring and discrete logarithm.”*

In our paper, we fill the mentioned definitional gap, and use the term “amortized” to evoke that the traditional anonymity requirements have purposely been weakened.

Further, we prove the security of our scheme based on the simple assumption that a zero-knowledge based signature scheme (e.g., any of [12], [21], [17], [30] ) is secure. (For concreteness, we present our scheme based on the Schnorr signature

scheme [30], whose security provably equals that of the discrete logarithm problem in the random-oracle model. Thus this security assumption alone suffices for amortized e-cash.)

We also attain efficiency in our scheme that is approximately as good as that in [26]. The actual efficiency issues will be discussed later, but for now we should mention that for simplicity we present a scheme in which only one coin can be spent at a time. This simplified scheme is not as efficient in terms of the computation needed to make and verify payments, but we present an improvement over the simplified scheme that makes our scheme comparable to divisible e-cash schemes in these aspects.

Finally, our algorithmic structure is conceptually simpler. For instance, rather than using complex tree-structures of special type of commitments, we can get by with conventional Merkle trees.

## 2 Definition of Wallet-Based E-Cash

### 2.1 Notation

PROTOCOLS.<sup>4</sup> A two-party protocol,  $P$ , to be run by parties  $A$  and  $B$ , is a pair of Interactive Turing Machines (ITMs):  $P = (P_A, P_B)$ . Following [14], on *input*  $(x, y)$ , where  $x$  is a private input for  $A$  and  $y$  a private input for  $B$ , and *random input*  $(r_A, r_B)$ , where  $r_A$  is a private random tape for  $A$  and  $r_B$  a private random tape for  $B$ , protocol  $(P_A, P_B)$  computes in a sequence of rounds, alternating between  $A$ -rounds and  $B$ -rounds. In an  $A$ -round ( $B$ -round) only  $A$  (only  $B$ ) is active and sends a message (i.e., a string) that will become an available input to  $B$  (to  $A$ ) in the next  $B$ -round ( $A$ -round). A computation of  $(P_A, P_B)$  ends in a  $B$ -round in which  $P_B$  sends the empty message and computes a private *output*.<sup>5</sup>

An ITM  $A$  is called a *polynomial-time* ITM (ptITM) if there exists a fixed polynomial  $p$  such that, for any ITM  $B$  and for any execution of  $(A, B)$  or  $(B, A)$  in which the length of  $A$ 's private input is  $\leq k$ , the number of steps taken by  $A$  in that execution is  $\leq p(k)$ . A *polynomial time protocol* is a protocol in which both ITMs are polynomial-time.

TRANSCRIPTS, VIEWS, AND OUTPUTS. Letting  $E$  be an execution of protocol  $(P_A, P_B)$  on input  $(x, y)$  and random input  $(r_A, r_B)$ , we make the following definitions:

- The *transcript* of  $E$  consists of the sequence of messages exchanged by  $A$  and  $B$ , and is denoted by  $\text{TRANS}^{P_A, P_B}(x, y, r_A, r_B)$ ;
- The *view of  $A$*  consists of the triplet  $(x, r_A, t)$ , where  $t$  is  $E$ 's transcript, and is denoted by  $\text{VIEW}_A^{P_A, P_B}(x, y, r_A, r_B)$ ;
- The *view of  $B$*  consists of the triplet  $(y, r_B, t)$ , where  $t$  is  $E$ 's transcript, and is denoted by  $\text{VIEW}_B^{P_A, P_B}(x, y, r_A, r_B)$ ;

<sup>4</sup> We shall use almost verbatim the protocol notation of [4].

<sup>5</sup> Due to the one-sidedness of secure computation, only machine  $P_B$  produces an output.

- The *output of E* consists of the string  $z$  output by  $P_B$  in the last round of  $E$ , and is denoted by  $\text{OUT}^{P_A, P_B}(x, y, r_A, r_B)$ .

We consider the three random variables  $\text{TRANS}(x, y, \cdot, r_B)$ ,  $\text{TRANS}(x, y, r_A, \cdot)$ , and  $\text{TRANS}(x, y, \cdot, \cdot)$ , respectively obtained by randomly selecting  $r_A$ ,  $r_B$ , or both, and then outputting  $\text{TRANS}(x, y, r_A, r_B)$ . We also consider the similarly defined random variables  $\text{VIEW}_A(x, y, \cdot, r_B)$ ,  $\text{VIEW}_A(x, y, r_A, \cdot)$ ,  $\text{VIEW}_A(x, y, \cdot, \cdot)$ ,  $\text{VIEW}_B(x, y, \cdot, r_B)$ ,  $\text{VIEW}_B(x, y, r_A, \cdot)$ ,  $\text{VIEW}_B(x, y, \cdot, \cdot)$ ,  $\text{OUT}_B(x, y, \cdot, r_B)$ ,  $\text{OUT}_B(x, y, r_A, \cdot)$ , and  $\text{OUT}_B(x, y, \cdot, \cdot)$ .

PROBABILISTIC EXPERIMENTS.<sup>6</sup> If  $A(\cdot)$  is an algorithm, then for any input  $x$ , the notation “ $A(x)$ ” refers to the probability space that assigns to the string  $\sigma$  the probability that  $A$ , on input  $x$ , outputs  $\sigma$ . The set of strings having a positive probability in  $A(x)$  will be denoted by “ $\{A(x)\}$ ”.

If  $S$  is a probability space, then “ $x \leftarrow S$ ” denotes the algorithm which assigns to  $x$  an element randomly selected according to  $S$ , and “ $x_1, \dots, x_n \leftarrow S$ ” denotes the algorithm that respectively assigns to  $x_1, \dots, x_n$ ,  $n$  elements randomly and independently selected according to  $S$ . If  $F$  is a finite set, then the notation “ $x \leftarrow F$ ” denotes the algorithm that chooses  $x$  uniformly from  $F$ .

If  $p$  is a predicate, the notation  $\text{Pr}[x \leftarrow S; y \leftarrow T; \dots : p(x, y, \dots)]$  denotes the probability that  $p(x, y, \dots)$  will be true after the ordered execution of the algorithms  $x \leftarrow S; y \leftarrow T; \dots$ .

The notation  $[x \leftarrow S; y \leftarrow T; \dots : (x, y, \dots)]$  denotes the probability space over  $\{(x, y, \dots)\}$  generated by the ordered execution of the algorithms  $x \leftarrow S, y \leftarrow T, \dots$ .

ADVERSARIAL TMS. An *adversarial TM* (ATM) is a probabilistic, polynomial-time Turing machine that is capable of retaining its internal state from one execution to the next. If the same ATM  $A$  occurs twice or more in a probabilistic experiment, it is understood that the first time  $A$  starts executing on its initial state, the second time starts with the state reached at the end of its first execution, and so on.

## 2.2 Wallet-Based E-Cash

To be as general as possible, a wallet-based e-cash scheme involves

- three players: the bank, the customer, and the merchant;
- three protocols: *withdrawal* (to be run between the bank and the customer to generate wallets and e-coins); *spending* (to be run between the customer and the merchant in which an e-coin is spent on a transaction); and *depositing* (to be run between the merchant and the bank, in which a merchant convinces the bank that an e-coin was spent with him); and
- a procedure *reveal* (in principle a protocol too), that reveals the customer’s identity in case of double spending.

<sup>6</sup> Verbatim from [2] and [15].

However, our scheme is simpler and more efficient: spending an e-coin consists of running an algorithm *pay* producing a signature-like string, which is universally verifiable by running an algorithm *ver*. Thus, assuming that a transaction specifies the merchant involved, depositing an e-coin simply consists of the merchant sending such a string to the bank (who can then run *ver* on its own). Further *rev* also is an algorithm, which reveals the identity of a double-spending customer when run on two payments relative to the same coin. It is therefore this better (because less interactive) type of wallet-based e-cash that we formalize below.

Because wallet-based e-cash is a special case of e-cash, our formalization will necessarily need to include basic properties common to other e-cash schemes. Because these are not particularly novel, however, we shall deal with them rather informally, so as to focus the reader's attention to the anonymity properties unique to wallet-based e-cash.

**Fundamental Components** A *wallet-based e-cash scheme* consists of

- A security parameter,  $1^k$  (i.e., an integer  $k$  in unary notation).
- A GMR-secure digital signature scheme,  $DSS = (GEN, SIG, VERIFY)$ . [15]  
(It is assumed that each DSS public key has been authenticated by a certifying authority.)
- A polynomial-time protocol  $W = (W_B, W_C)$ , for e-coin withdrawal.
- A probabilistic polynomial-time algorithm *pay*.
- A probabilistic polynomial-time algorithm *ver*.
- A probabilistic polynomial-time algorithm *rev*.

**Basic Properties** *Honest inputs and honest outputs.* Protocol  $W$  is performed between the bank and the customer. In an execution of  $W$ , both parties must be aware of  $n$ , the size of the wallet to be produced, and the security parameter  $1^k$ . The bank takes as additional inputs its DSS secret key,  $SK_B$ , and the DSS public key of the customer,  $PK_C$ . The customer takes as an additional input his DSS secret key,  $SK_C$ .

Because  $W_C$  is the second ITM in protocol  $W$ , it is the customer who sees the output of  $W$ . Each such output is called a *wallet* and consists of a pair  $(x, x_s)$ . Component  $x$  is referred to as the *public wallet* (since it is shown with payment) and specifies its own size,  $n$ , and the public key,  $PK_B$ , of the bank that issued it. Component  $x_s$  is referred to as the *secret wallet*, since it must be kept secret as it allows the customer to spend the coins in the wallet. Conceptually,  $x$  and  $x_s$  can be considered as a public-secret key pair of a special kind of digital signature scheme.<sup>7</sup> It is worth remarking that  $x$  and  $x_s$  *together* make up the common subcoin for any coin in the wallet, while the individual subcoin is just an integer  $i$  in the range  $[1, n]$ .

Algorithm *pay* receives 4 inputs:  $x, x_s$ , a wallet of size  $n$ , an integer  $i \in [1, n]$  (the coin number), and  $t$ , the transaction on which to spend the coin

<sup>7</sup> It is not quite a signature key pair, because it has not been generated by an independent key generation algorithm, but has similar properties.

(conceptually, a string to be signed). The output of *pay* on such inputs is a 4-tuple  $y, x, i, t$ . Conceptually,  $y$  is the signature of  $t$  with public key  $x$  and coin  $i$ .

Algorithm *ver* receives 4 inputs:  $y, x, i$ , and  $t$ , and outputs a bit: 1 indicates that the payment was valid, while 0 indicates otherwise.

If the customer and the bank behave honestly in withdrawal and payment, the customer should be able to spend any coin in the wallet produced. That is, if *ver* is run on the output of an honest execution of *pay*, based on a wallet of size  $n$  which was the output from an honest execution of *W*, and a value  $i \in [1, n]$ , *ver* should always output 1.

*E-coins cannot be forged.* Informally, this is the old e-cash property that a malicious customer cannot ever spend more different coins than he withdrew from the bank. In our setting, this means two things. First, he cannot interact with an honest bank and end up with a wallet with more than  $n$  coins if the common input was  $n$ . Second, he cannot use a wallet of size  $n$  to produce a valid payment on a coin  $i > n$ . This basic property remains true even if the customer is allowed to be adaptive, in that he may interact with the bank to withdraw wallets many times, each time based on information he learned before.

*Double-spending and no framing.* On any input, algorithm *rev* outputs either the empty string  $\varepsilon$  or a customer identity with the following constraints:

1. On input two quadruples  $(y_1, x, i, t_1)$  and  $(y_2, x, i, t_2)$  such that  $ver(y_1, x, i, t_1) = ver(y_2, x, i, t_2) = 1$ , but  $t_1 \neq t_2$ , algorithm *rev* outputs the identity of some customer with probability essentially equal to 1.
2. If a customer  $C$  never double spends any coins, then it is computationally hard to find an input to *rev* (even given all the payments  $C$  legitimately makes) on which *rev* outputs  $C$ 's identity.

This, we can consider inputs to *rev* that produce a customer's identity to be irrefutable proof that the customer has double-spent, since if they have not, it would be computationally intractable to generate such proof. (It should be noted that while this requirement is so formulated not to lose generality, in our scheme a "more convincing" requirement is actually met: namely, *rev* outputs a customer's DSS signature of the sentence "I have double spent.")

*No spending on behalf of others.* If  $(x, x_s)$  is a wallet of an honest customer  $C$ , then it is computationally hard to generate a 4-tuple  $(y, x, i, t)$  such that  $ver(y, x, i, t) = 1$  with only  $x, i$ , and  $t$  as inputs. This continues to hold even if the adversary can impersonate any player except  $C$  and force  $C$  to withdraw arbitrarily many wallets of arbitrary size in an adaptive manner, as well as produce payment for arbitrarily many transactions  $t' \neq t$  in an adaptive manner.

**Anonymity** We now wish to express more formally the two crucial anonymity properties, that characterize wallet-based e-cash, that we discussed informally in the introduction.

*Wallet anonymity.* At a minimum, this requirement should ensure that given a *single* coin  $i$ , from some wallet  $x$ , it is impossible to determine to which customer the coin belongs. Notice that the customer necessarily is one having withdrawn money. Moreover, because each coin reveals the size of the wallet, the customer must be one who has retrieved a wallet of that size. Thus, wallet anonymity guarantees that, if  $m$  customers have withdrawn wallets of size  $n$ , the bank cannot determine which of the customers is spending a given coin better than random guessing. This should remain true even if the bank is dishonest both during its own key generation (it may help to have a special key rather than a random one!) and during the withdrawal protocol.

But we wish to guarantee a stronger property, namely, that the customer behind *any* spent coin cannot be guessed better than at random, even if the bank has available how all other  $mn - 1$  coins have been spent. This should be true even if the bank chooses all the transactions on which each coin is to be spent (which might arise via collaboration with the merchants). We allow these transactions to be chosen adaptively as follows. Initially, the  $m$  customers' original names are renamed according to a random permutation  $\sigma: i \rightarrow \sigma(i)$ . After that, the bank chooses a transaction  $t_1$ , a customer name  $j_1$  (which under the permutation corresponds to customer  $\sigma(j_1)$ ), and a coin  $i_1$  from that customer's wallet. Then, the bank receives a payment of that transaction with that coin. Based on this result, the bank chooses a second transaction, a second identity, and a second coin, and receives the corresponding payment. And so on, for  $mn$  times. At the end, we let the bank choose a single coin and guess the *original* identity of its customer. Of course, if the bank asks for any coin to be spent twice, it will (by double-spending) learn the identity of the corresponding customer, but if all spending requests relate to different coins, then the bank should not be able to do better than random guessing.

Formally, denoting by  $S_m$ , the set of all permutations over  $m$  elements, wallet anonymity is so expressed:

$$\begin{aligned}
& \forall c > 0, \forall m, n > 0, \forall \text{ATM } A, \exists k_0 : \forall k > k_0 \\
& Pr[(SK_{C_1}, PK_{C_1}), \dots, (SK_{C_m}, PK_{C_m}) \leftarrow \text{GEN}(1^k); \\
& \quad (SK_B, PK_B) \leftarrow A(PK_{C_1}, \dots, PK_{C_m}); \\
& \quad (x_1, x_{s_1}) \leftarrow \text{OUT}^{A, W_C}(\varepsilon, (SK_{C_1}, PK_B, n), \cdot, \cdot); \\
& \quad \dots \\
& \quad (x_m, x_{s_m}) \leftarrow \text{OUT}^{A, W_C}(\varepsilon, (SK_{C_m}, PK_B, n), \cdot, \cdot); \\
& \quad \sigma \leftarrow S_m; \\
& \quad (t_1, j_1, i_1) \leftarrow A(\varepsilon); \\
& \quad (y_1, x_{\sigma(j_1)}, i_1, t_1) \leftarrow \text{pay}(x_{\sigma(j_1)}, x_{s_{\sigma(j_1)}}, i_1, t_1); \\
& \quad (t_2, j_2, i_2) \leftarrow A(y_1, x_{\sigma(j_1)}); \\
& \quad \dots \\
& \quad (y_{mn}, x_{\sigma(j_{mn})}, i_{mn}, t_{mn}) \leftarrow \text{pay}(x_{\sigma(j_{mn})}, x_{s_{\sigma(j_{mn})}}, i_{mn}, t_{mn});
\end{aligned}$$

$$(a, b) \leftarrow A(y_{mn}, x_{\sigma(j_{mn})}) : \\ \sigma(a) = b \text{ and } ((j_u, i_u) = (j_v, i_v) \Rightarrow u = v) < 1/m + k^c.$$

*Wallet unlinkability.* At a minimum, this requirement should ensure that given a pair of wallets each of the same size, it is impossible for the dishonest bank to determine whether the wallets belong to the same customer or not. This is extended and formalized by means of the following game.

First, each of the two customers withdraws two wallets (each consisting of  $n$  coins).

Second, the bank determines adaptively the transactions on which all the  $4n$  coins will be spent. Initially, the 4 original wallets are renamed according to a random permutation  $\sigma: i \rightarrow \sigma(i)$ . Then, the bank selects a transaction  $t_1$ , and a wallet name  $j_1$  (which under the permutation corresponds to wallet  $\sigma(j_1)$ ) as well as a coin  $i_1$  in that wallet. The bank then receives a payment of that transaction with that coin from customer who owns wallet  $\sigma(j_1)$ . (Notice that such a customer is  $\lfloor \sigma(j_1)/2 \rfloor$ , if we assume that customer 0 owns wallets 0 and 1, and customer 1 owns wallets 2 and 3.) Based on this result, the bank chooses a second transaction, a second (possibly different) wallet, and a second coin, and receives the corresponding payment, and so on, for  $4n$  times.

Finally, we let the bank choose two different wallets: the bank tries to choose two wallets that belong to the same customer. Of course, if the bank requests double spending, it will be able to correctly choose two such wallets, but if all spending requests relate to different coins, then the bank should not be able to do better than randomly guessing a pair of wallets (which has a probability of success of  $1/3$ , since there are two correct answers out of six choices). Formally, wallet unlinkability is so expressed:

$$\forall c > 0, \forall n > 0, \forall \text{ATM } A, \exists k_0 : \forall k > k_0 \\ Pr[(SK_{C_0}, PK_{C_0}), (SK_{C_1}, PK_{C_1}) \leftarrow \text{GEN}(1^k); \\ (PK_B, SK_B) \leftarrow A(PK_{C_0}, PK_{C_1}); \\ (x_0, x_{s_0}) \leftarrow \text{OUT}^{A, W_C}(\varepsilon, (PK_B, SK_{C_0}, n), \cdot, \cdot); \\ (x_1, x_{s_1}) \leftarrow \text{OUT}^{A, W_C}(\varepsilon, (PK_B, SK_{C_0}, n), \cdot, \cdot); \\ (x_2, x_{s_2}) \leftarrow \text{OUT}^{A, W_C}(\varepsilon, (PK_B, SK_{C_1}, n), \cdot, \cdot); \\ (x_3, x_{s_3}) \leftarrow \text{OUT}^{A, W_C}(\varepsilon, (PK_B, SK_{C_1}, n), \cdot, \cdot); \\ \sigma \leftarrow S_4; \\ (t_1, j_1, i_1) \leftarrow A(\varepsilon); \\ (y_1, x_{\sigma(j_1)}, i_1, t_1) \leftarrow \text{pay}(x_{\sigma(j_1)}, x_{s_{\sigma(j_1)}}, i_1, t_1); \\ (t_2, j_2, i_2) \leftarrow A(y_1, x_{\sigma(j_1)}); \\ \dots \\ (y_{4n}, x_{\sigma(j_{4n})}, i_{4n}, t_{4n}) \leftarrow \text{pay}(x_{\sigma(j_{4n})}, x_{s_{\sigma(j_{4n})}}, i_{4n}, t_{4n}); \\ (a, b) \leftarrow A(y_{4n}, x_{\sigma(j_{4n})}) : \\ a \neq b \text{ and } \lfloor \sigma(a)/2 \rfloor = \lfloor \sigma(b)/2 \rfloor \text{ and } ((j_u, i_u) = (j_v, i_v) \Rightarrow u = v) < 1/3 + k^c.$$

### 3 Our Scheme

Our solution relies on zero-knowledge based signature schemes such as those presented in [12], [21], [17], [30]. Such schemes sign a message  $M$  in three steps: a *commitment step*, which is independent of the message, a *challenge step*, which is message dependent, and a *response step*, in which the secret key is used. Our e-cash scheme can work with *any* such a signature scheme. But, for concreteness (and to avoid an ad hoc, general notation), in this extended abstract we shall present our scheme based on the Schnorr signature scheme. Our solution, like most other e-cash schemes, relies on blind signatures [5] for security.

#### 3.1 Blind Signatures

We present an ad hoc summary of blind signatures, using blind RSA signatures as our example [5].

In a blind signature scheme, there is a finite message space, and if Alice wants Bob to sign a message  $M$  without revealing it to him, she first generates a random value  $\rho$  and then uses the “blinding” function  $F$  to compute a random  $M' = F(M, \rho)$  and asks Bob to generate the signature  $\text{SIG}_{SK_B}(M')$  on  $M'$ . (Function  $F$  is such that, for every  $M$ , for a random choice of  $\rho$   $F(M, \rho)$  is uniformly distributed in the message space.) Then, Alice uses the “unblinding” function  $G$  to compute  $\text{SIG}_{SK_B}(M) = G(\text{SIG}_{SK_B}(M'), \rho)$ . For example, suppose  $N = PQ$  is an RSA public modulus and that  $d$  and  $e$  are the private and public exponents, respectively, for Bob’s RSA key. In order to compute a blind signature, Alice generates  $\rho$ , a random value modulo  $N$ , and computes  $M' = F(M, \rho) = \rho^e M$ . Then, Bob computes the signature  $(\rho^e M)^d = \rho M^d$  and Alice computes  $M^d$  by dividing  $\rho M^d$  by  $\rho$  modulo  $N$ . Conceptually, Bob will have no idea what he signed since the distribution of  $M'$  is independent of  $M$ . To highlight that any secure blind signature scheme can be used with our protocol, we will use the abstract notation. However, it should be mentioned that it is important if the overall scheme is to have security based on only one cryptographic assumption that a blind signature scheme be chosen whose security depends on the same assumption. For example, the protocol of [20] depends on the discrete logarithm assumption, which is in line with the other assumptions we will need (since we use the Schnorr signature scheme).

#### 3.2 Schnorr’s Signature Scheme

We present a summary of the Schnorr signature scheme [30]:

- **Common public parameters:**  $p$  and  $q$  (two large primes such that  $q$  divides  $p-1$ ),  $g$  (a generator of the subgroup of  $Z_p^*$  of size  $q$ ), and  $H$  (a collision resistant hash function producing outputs in  $Z_q$ ).
- **Secret-public key pairs:**  $(x, g^x)$ , where  $x$  is generated at random in the interval  $[1, q-1]$ .

- **Signing a message  $M$ :**
  - STEP 1: Randomly select  $r \in [1, q - 1]$  and compute  $g^r \bmod p$  (the commitment)
  - STEP 2: Compute  $e = H(M, g^r)$  (the challenge)
  - STEP 3: Output  $(g^r \bmod p, r + ex \bmod q)$  (the signature of  $M$ )
- **Verifying a signature of a message  $M$ :** To verify a signature  $(a, b)$ , compute  $e = H(M, a)$  and check that  $g^b = a(g^x)^e \bmod p$ .

### 3.3 Crucial Properties of the Schnorr Scheme

The quantity  $g^r \bmod p$  computed in Step 1 is called an *ephemeral key*, because it is generated just as a public key and because it is used only once: signing a new message entails generating a new ephemeral key.

Let us now highlight two properties of the Schnorr scheme that we use heavily in our scheme:

1. *The ephemeral key can be generated off line.* That is, it can be generated before the message to be signed is chosen or known:  $g^r$  is a random public key!
2. *If the ephemeral key is fixed, then the Schnorr scheme is strictly one-time.* That is, if one insists in using the Schnorr scheme only with a particular ephemeral key, then the scheme is secure (i.e., not existentially forgeable against a chosen ciphertext attack) as long as one signs no more than one message. However, as soon as one signs any two different messages  $M_1$  and  $M_2$  with the same ephemeral key, then the scheme becomes totally insecure, meaning that the very secret signing key  $x$  is revealed. Indeed, because  $H$  is collision resistant and because  $M_1 \neq M_2$ , we are essentially guaranteed that  $e_1 = H(M_1, g^r) \neq e_2 = H(M_2, g^r)$ , and the two signatures will consist of  $g^r \bmod p$  and, respectively,  $r + e_1x$  and  $r + e_2x \bmod q$ . Because  $e_1 - e_2 \neq 0 \bmod q$ , we can solve:

$$x = [(r + e_1x) - (r + e_2x)](e_1 - e_2)^{-1} \bmod q.$$

3. *The Schnorr scheme's security is based on simple assumptions.* The security of the Schnorr scheme depends on the difficulty of the discrete logarithm problem. The proof of this is done under the random oracle model in the paper by Schnorr [30].

### 3.4 Our Use of the Crucial Properties

To embed the customer's identity we use a secure symmetric scheme  $(E', D')$ . (Our scheme actually construct  $(E', D')$  from an underlying secure symmetric encryption scheme  $(E, D)$ . The precise description of how  $(E', D')$  are derived from  $(E, D)$  and why this auxiliary step is needed will be explained in the security sketch, section 3.5.)

Our scheme also uses Schnorr’s signature scheme with public parameters  $p, q, g$ , and  $H$ , but only to compute and verify payments. (If the security parameter is  $1^k$ , then we assume  $q$  is of length  $k$  and thus so are the outputs produced by  $H$ .)

For all other purposes, as demanded by our definition, our scheme uses a second signature scheme. To avoid confusion between the two schemes, the Schnorr keys and signatures are explicitly spelled out, while keys and signatures under the second scheme are denoted “abstractly.” Thus, the permanent key pair (i.e., that relative to the second scheme) of a customer  $C$  is denoted  $(PK_C, SK_C)$ , and that of a bank  $B$  is denoted  $(PK_B, SK_B)$ . Customer  $C$ ’s signature of a message  $M$  in this second scheme will be denoted by  $SIG_{SK_C}(M)$ ;  $B$ ’s signature of  $M$  will be denoted by  $SIG_{SK_B}(M)$ .

NOTE: For simplicity, we assume that in the second scheme the signature of a message  $M$  always includes  $M$  (in the clear).

The second signature scheme is chosen so as to allow blind signatures [5]. To highlight that any such a secure scheme can be used, we use the following abstract notation. There are two efficient algorithms,  $F$  and  $G$ . The bank’s blind signature of  $M$  is obtained by (1) computing  $F(M) = (M', \rho)$ ; (2) asking the bank to sign  $M'$ , and (3) computing  $SIG_{SK_B}(M)$  by running  $G(SIG_{SK_B}(M'), \rho)$ .

**The withdrawal protocol  $W$ .** For simplicity, we assume that each wallet size is a fixed power of 2,  $2^d$ .

Here is how a customer withdraws a wallet of size  $2^d$ .

1. The customer generates a random public/private key pair  $(g^x, x)$  in the Schnorr scheme.

2. The customer uses his permanent secret key to sign the sentence “ $C$  has double spent”; that is, he computes  $s = SIG_{SK_C}(C \text{ has double spent})$ . Then, he uses the symmetric encryption scheme to encrypt signature  $s$  using as encryption key the just generated Schnorr secret signing key; that is, he computes  $z = E'_x(s)$ .

3. The customer then generates  $2^d$  ephemeral key pairs  $(g^{r_1}, r_1) \dots (g^{r_{2^d}}, r_{2^d})$ .

4. The customer creates a Merkle hash tree [Mer80],  $T$ , of depth  $d$  which stores the public parts of the ephemeral keys  $(g^{r_1}, \dots, g^{r_{2^d}})$  in its leaves (where  $g^{r_i}$  is stored in leaf  $i$ ). Denote by  $R$  the root value of  $T$ .

5. The customer generates a random value  $\rho$ , computes  $M' = F(M, \rho)$ , and sends  $M'$  to the bank, where  $M = (R, d, z, g^x)$ .

6. The customer proves interactively with the bank that  $M'$  has properly formed. Specifically, the customer proves that  $M' = F(M, \rho)$  where  $M$  is a quadruple of values  $(a_1, a_2, a_3, a_4)$  such that (1)  $a_2 = d$  and (2)  $a_3$  is the encryption under  $E'$ , using the discrete log of  $a_4$  as a secret key, of  $C$ ’s permanent signature of the value “ $C$  has double spent”. These proofs are accomplished via general zero-knowledge proof methods, such as those in [16].<sup>8</sup>

<sup>8</sup> The [16] result shows how to solve any NP problem in zero-knowledge. The problem of whether  $M'$  was generated from appropriate inputs can obviously be solved in NP (we just “guess” the  $a_1, a_2, a_3, a_4$  values that were used).

(*Comment:* the complexity of such a zero-knowledge proof is essentially independent of  $d$ : it grows in  $\text{poly}(\log(d))$ , and thus is polynomial in  $\log \log(n)$ , where  $n = 2^d$  is the size of the wallet.)

7. The bank provides a signature  $\text{SIG}_{SK_B}(M')$  and sends it to the customer.

8. The customer unblinds the signature, that is, he computes  $w = G(\text{SIG}_{SK_B}(M'), \rho)$ . The public part of the wallet consists of  $(w, R, d, z, g^x)$ , while the secret part of the wallet consists of the secret parts of the ephemeral keys  $(r_1, \dots, r_{2^d})$  along with the secret key  $x$ .

(*Comment:* the public wallet is compact as its size does not depend on  $d$ , but on  $\log d$ . As described above, the secret wallet grows linearly in  $n$ , the size of the wallet. However,  $r_1, \dots, r_{2^d}$  could be outputs from a pseudorandom generator on input a short seed of length  $k$ , the security parameter: then, the secret wallet could consist of just  $2k$  bits: the seed and  $x$ .)

**The payment algorithm  $\text{pay}$ .** On input  $((w, R, d, z, g^x), (r_1, \dots, r_{2^d}), x, i, t)$ ,  $\text{pay}$  runs as follows:

1. The customer (re)generates the ephemeral public key  $g^{r_i}$  from  $r_i$  and the public parameters. Then he computes  $(g^{r_i} \bmod p, r_i + ex \bmod q)$ , the Schnorr signature on message  $t$  using secret key  $x$  and ephemeral key pair  $(g^{r_i}, r_i)$ .

2. The customer generates the authentication path,  $\text{path}$ , in Merkle tree  $T$  for leaf  $i$ . (*Comment:* This list of  $d$  values of length  $k$  authenticates  $g^{r_i}$  as the  $i$ th public ephemeral key.)

3. The customer outputs  $((g^{r_i} \bmod p, r_i + ex \bmod q, \text{path}), (w, R, d, z, g^x), i, t)$

**The payment verification algorithm  $\text{ver}$ .** On input  $((a, b, \text{path}), (w, R, d, z, g^x), i, t)$ ,  $\text{ver}$  runs as follows:

1. Check that  $(a, b)$  is a valid Schnorr signature relative to public key  $g^x$  on message  $t$ . If not, halt and output 0.

2. Check that  $w$  is a signature on  $(R, d, z, g^x)$  relative to the bank's public key  $PK_B$ . If not, halt and output 0.

3. Check that  $\text{path}$  is of size  $d$  and correctly authenticates that  $a$  is stored in the  $i$ th leaf of a Merkle tree with root value  $R$ . If not, output 0.

4. (Else) output 1.

**The identity revealing algorithm  $\text{rev}$ .** The  $\text{rev}$  algorithm, on input  $y_1 = ((a_1, b_1, \text{path}_1), (w_1, R_1, d_1, z_1, g^{x_1}), i_1, t_1)$  and  $y_2 = ((a_2, b_2, \text{path}_2), (w_2, R_2, d_2, z_2, g^{x_2}), i_2, t_2)$  runs as follows:

1. Check that (1)  $\text{ver}(y_1) = 1$ , (2)  $\text{ver}(y_2) = 1$ , (3)  $w_1 = w_2$ , (4)  $i_1 = i_2$ , and (5)  $t_1 \neq t_2$ . If any of these checks fail, halt and output  $\varepsilon$ .

2. Let  $e_1 = H(t_1, a_1)$  and  $e_2 = H(t_2, a_2)$ , and compute  $(e_1 - e_2)^{-1} \bmod q$ .

3. Compute  $x = (b_1 - b_2)(e_1 - e_2)^{-1} \bmod q$ .

4. Compute  $s = D'_x(z_1)$ . Output  $s$ , which should be customer  $C$ 's signature of "C has double spent."

### 3.5 Security Sketch

In this extended abstract, we only informally give arguments that our scheme satisfies our definitions of a wallet-based e-cash scheme. In particular,

*Honest inputs and honest outputs.* All the algorithms fit the format specified for a wallet-based e-cash scheme. All that must be shown is that when the withdrawal protocol is run honestly, the customer can successfully spend any coin in the wallet she generates.

Since the customer has access to  $x, r_i$ , and  $t$ , she can generate the signature  $(g^{r_1} \bmod p, r_i + ex \bmod q)$  correctly. Furthermore, since she knows all the leaves of the Merkle tree, she can easily compute  $path$ . Thus, she can compute  $pay$  correctly.

Since the signature generated in  $pay$  was valid, and the wallet involved was signed by the bank in the withdrawal protocol, and  $path$  was correct, the algorithm  $ver$  will approve the output the customer generates.

*E-coins cannot be forged.* Since we assume that the blind signature scheme is immune to forgery, and the zero-knowledge proofs are correctly implemented, there is no way the customer can create any wallets other than those she generates with the bank, and the wallets so generated all have the same size as the bank was aware of. Thus, the only possible way the customer could forge coins is if the customer were able to spend more than  $2^d$  different coins out of a wallet. However, if this were possible, the customer would have been able to produce  $2^d + 1$  different authentication paths of length  $d$ , which can be used to produce a hash collision. Since we assume it is computationally hard to find hash collisions, it is computationally hard for the customer to forge coins.

*Double-spending and no framing.* Clearly, if the customer spends the same coin on two different transactions, then unless those two transactions represent a collision for the hash function  $H$ , then the  $rev$  algorithm “extracts”  $x$  and decrypts  $z$  to get the user’s identity. On the other hand, since the Schnorr signature scheme is secure when we never use an ephemeral key more than once, it is computationally hard for the bank to come up with inputs to  $rev$  which reveal a customer’s identity if that customer did not double spend any coins.

*No Spending on behalf of others.* If one has a wallet but is not aware of  $x$ , one cannot produce signatures where  $x$  is the secret signing key, because the Schnorr signature scheme is secure. Thus, one cannot sign without access to the secret wallet.

*Wallet anonymity.* Informally, in the withdrawal protocol the bank only sees  $M'$  (which it receives in Step 5 and digitally signs by  $w$ ), and the values  $PK_C$  and  $d$  (which it receives in order to carry out the zero knowledge proof that  $M'$  is the “blinded version” of a wallet being issued to customer  $C$  of size  $2^d$ ).

The bank never sees  $M = (R, d, z, g^x)$ . Moreover, by the security of a blind signature, the bank cannot infer anything about  $M$  from  $M'$ . It is true that the

customer proves to the bank that  $M'$  corresponds to an  $M$  which embeds an encryption of the customer's identity. But because this proof is a zero-knowledge one, the bank cannot learn any more about  $M$  than claimed.

During payment verification, the bank only sees the values  $((a, b, path), (w, R, d, z, g^x), i, t)$ , the value  $M = (R, d, z, g^x)$ , its own signature  $w$  of  $M$ , the coin number  $i$ , the transaction  $t$ , the Schnorr's signature  $(a, b)$  of  $t$  relative to public key  $g^x$ , and the authentication path that coin  $i$  equals  $a$  in the Merkle tree rooted at  $R$ . Thus the only thing the bank learns which depends on the customer's identity in any way is  $M$ , because  $M$  contains the value  $z = E'_x(\text{SIG}_{SK_C}(C \text{ has double spent}))$ . Due to the security of the blind signature, the bank cannot link  $M$  to the  $M'$  it saw during withdrawal. Thus, the only way the bank can associate the customer's identity to his payments is through  $z$ , and all that remains to prove is that this is impossible.

Being the encryption scheme  $E'$  secure, if  $\text{SIG}_{SK_C}(C \text{ has double spent})$  were encrypted with a *totally secret* random key,  $z$  would provably not betray  $C$ . Notice, however, that, in order to guarantee that double spending reveals  $C$ , we must encrypt  $\text{SIG}_{SK_C}(C \text{ has double spent})$  with key  $x$ , that is with the same secret key corresponding to the Schnorr's public key  $g^x$  that is known to the bank. While in practice this step may very well be secure, the same cannot be claimed from a theoretical point of view:  $x$  is essentially random (because the secret key in the Schnorr scheme is a random element of  $Z_q$  and therefore is essentially a random string), but it is not totally secret (because  $g^x$  is information about  $x$ ). The best way to prove that  $g^x$  does not interfere with the security of encryption would be a "simulation argument," but no such an argument appears to be possible here. Therefore, we resort to design the encryption scheme  $E'$  so that we can prove that the information provided by  $g^x$  is computationally irrelevant.

The idea is to use the Goldreich-Levin bit construction [GL89].

Let us assume that the discrete logarithm problem is at least as hard as  $2^{k^\alpha}$ . More formally,

$$\begin{aligned} &\forall A \text{ probabilistic TM } \exists n : \forall k \geq n \\ &Pr[(q, g) \leftarrow \text{DLGEN}(1^k); x \leftarrow \{2, \dots, q-2\}; \\ &y \leftarrow A_{2^{k^\alpha}}(q, g, g^x) : y = x] < \frac{1}{2^k}, \end{aligned}$$

where  $A_{2^{k^\alpha}}(q, g, g^x)$  denotes that  $A$  is allowed to run on input  $q, g, g^x$  for only  $2^{k^\alpha}$  steps before its execution is cut off and an empty result returned.

This is not the traditional discrete logarithm assumption. The difference is that in the traditional discrete logarithm assumption, we assume the above is true for *some*  $\alpha > 0$ , but in our case we need to use  $\alpha$  in the protocol, so we need to assume the truth of the above statement using our given  $\alpha$ .

Let  $f : \{0, 1\}^\ell \rightarrow \{0, 1\}^\ell$  be a one-way function with complexity greater than  $2^k$ , let  $x$  be a random  $\ell$ -bit string, and let  $\rho_1, \dots, \rho_k$  be  $\ell$ -bit random strings. Then informally, on auxiliary inputs  $f, f(x)$  and  $\rho_1, \dots, \rho_k$ , the  $k$  bits  $b_i = x \cdot \rho_i$  are indistinguishable from  $k$  random bits by any polynomial-time algorithm.

This suggests to construct  $(E', D')$  from a secure symmetric cipher  $(E, D)$  and Schnorr public parameters  $p, q$  and  $g$  as follows. Select  $\rho_1, \dots, \rho_k, x$  at random in  $Z_q$ , compute  $g^x \bmod p$ , compute  $b_1, \dots, b_k$  such that  $b_i =$

$x \cdot \rho_i$ , and define the encryption of a string  $\sigma$  with key  $x$  to be  $E'_x(\sigma) = (g^x \bmod p, \rho_1, \dots, \rho_k, E_{b_1 \dots b_k}(\sigma))$ . Because  $b_1 \dots b_k$  is polynomially random even given knowledge of  $g^x \bmod p, \rho_1, \dots, \rho_k$ , this encryption is secure.

In terms of how to apply these techniques to our protocol, if  $k$  is the security parameter desired for the scheme, it should represent the length of the encryption key. Thus, the security parameter needed for the Schnorr scheme is in fact not  $k$  but  $k^{1/\alpha}$ . (It is presumed that  $0 < \alpha < 1$ ). The encryption of a message  $M$  is computed by first generating  $k$  random bit strings  $t_1, \dots, t_k$  of length equal to that of  $x$ . (The discrete log function  $x \mapsto g^x$  is the one-way function we will use.) Then, the bits  $b_1, \dots, b_k$  are calculated by computing  $b_i = x \cdot t_i$ . Finally, the underlying secure symmetric encryption scheme is used to compute  $E_b(M)$  where  $b = b_1 b_2 \dots b_k$ . The ciphertext consists of the list of strings  $t_1, \dots, t_k$  and the encryption  $E_b(M)$ . Decryption using  $x$  as the key involves recalculating  $b$  and then decrypting  $C = E_b(M)$  to get  $M$ .

For our scheme to be secure, we depend on the strong discrete log assumption described above. It is worth mentioning, however, that in practice the difference between this assumption and the discrete log assumption is insignificant. Whenever we use a scheme the security of which is based on some problem, we must make a concrete guess as to the difficulty of that problem. Without doing this, we could not adequately choose security parameters to use in such schemes. Thus, ultimately, our scheme differs from these other schemes only in that we use this guess about the difficulty of the discrete logarithm problem not just to pick our security parameter but also in the details of our scheme.

*Wallet unlinkability.* Since the scheme satisfies wallet anonymity, the only way an adversary can correlate two wallets of the same customer is for payments from those two wallets to be correlated to one another.

Informally, the only way in which two different wallets belonging to the same customer are related is that the same customer's identity is embedded in both wallets. Other than these two encryptions,  $z_1$  and  $z_2$ , all other information in the two wallets is generated independently.

Thus, if the symmetric encryption algorithm satisfies polynomial indistinguishability, to determine whether  $z_1$  and  $z_2$  relate to the same customer is computationally hard.

## 4 Efficiency

We are concerned with the time complexity of the withdrawal protocol, the storage requirement of the wallet, the time complexity of the *pay* algorithm, and the time complexity of the *ver* algorithm. In the following, we will assume  $1^k$  is the security parameter, and  $n$  is the number of coins in a wallet. *poly* will represent some unnamed polynomial.

The withdrawal protocol requires  $O(n)$  modular exponentiations (modulo a prime the size of which depends on  $1^k$ ) which the customer computes before interacting with the bank, a zero-knowledge proof protocol between the bank

and the customer, which takes  $O(\text{poly}(\log \log n + 1^k))$  time, and a signature by the bank on a message of length  $O(\log \log n + 1^k)$ . Thus, the entire protocol takes time  $O(\text{poly}(\log \log n + 1^k))$ .

There is a simple trade-off between the time complexity of the *pay* algorithm and the storage size of a wallet. On one side, the customer keeps stored the entire tree and all its leaves (which takes  $O(1^k n)$  space), in addition to  $x$  and  $d$ , but needs only compute one modular exponentiation in *pay*. On the other side, the customer keeps only  $x, d$  and a pseudo-random seed stored in the wallet, which takes only  $O(1^k + \log \log n)$  space, but must compute all the leaves each time, which takes  $O(n)$  modular exponentiations.

However, there is a middle ground. The customer can choose a parameter  $\epsilon$  (presumably as large as possible so that his available storage space is not exceeded), and store the top  $h = \epsilon \log n$  levels of the tree in addition to  $x$  and  $d$ . This takes  $O(1^k n^\epsilon)$  storage. The customer will still have to recompute part of the tree each time they execute the *pay* protocol, but the portion they must compute is smaller, and thus requires only  $O(n^{1-\epsilon})$  modular exponentiations.

The time complexity of the *ver* algorithm is simply the time required to do two signature verifications.

#### 4.1 Multi-Coin Spending

It is also worth discussing a solution to the problem that the scheme described above is only capable of spending one coin at a time. However, it can be easily modified to spend multiple coins at once as follows.

Instead of having a typical hash tree where the parent is computed from left and right children by computing  $H(LR)$  where  $LR$  denotes the concatenation of the two child node values, we each internal node of the tree have its own ephemeral key  $g^r$ . Then, the parent node is computed as  $H(Lg^r R)$ . In order to spend a node (and any node may be spent, not just leaf nodes) the customer must provide a signature on the transaction  $t$  using the ephemeral key of that node, and must provide a signature on a standard phrase (for example, "PATH") on all ancestors of that node. The customer must also provide the ephemeral keys for all internal nodes on the path from the node being spent to the root.

Now, in order to prevent double-spending we must ensure that (1) no node can be spent twice without revealing the customer's identity and that (2) if a node is spent, no ancestor or descendent of that node can be spent without revealing the customer's identity.

If the customer violates (1), then some node will be used to sign two different transactions  $t$  and  $t'$  and these signatures can be used to reveal  $x$ .

If the customer violates (2), then the node of the two which was closer to the root would be used to sign both the standard phrase and a transaction  $t$ , and these signatures can be used to reveal  $x$ .

Note that two different descendents of the same node can be spent without revealing the customer's identity. Those keys on both paths will only be used to sign the standard phrase, and thus the bank cannot recover the customer's secret key. Also note that the customer does not actually have to take the time

to generate all the signatures on the standard phrase when running the spending protocol; these can be precomputed and stored with the secret information of the wallet.

With this technique, transactions of size  $m$  can be completed by spending  $O(\log m)$  nodes in this manner. This requires  $O(d \log m)$  signature verifications for the total transaction, which may be more than  $m$  (for example, if  $m$  is small relative to  $d$ ). The size of the wallet also grows by a factor of 2.

## 5 Acknowledgements

The authors would like to acknowledge valuable discussions with Madhu Sudan which led to the development of the multi-coin spending technique described in section 4.1

## References

- [1] S. Brands. Untraceable off-line cash in wallet with observers. In *Advances in Cryptology – CRYPTO ’93*, 1993.
- [2] M. Blum, A. De Santis, S. Micali, and G. Persiano. Noninteractive Zero-Knowledge. In *SIAM Journal on Computing* 20(6): pp. 1084-1118, 1991.
- [3] E. Brickell, P. Gemmell, and D. Kravitz. Trustee-Based Tracing Extensions to Anonymous Cash and the Making of Anonymous Change. In *Proceedings of SODA ’95*, 1995.
- [4] A. Beimel, T. Malkin, and S. Micali. The All-or-Nothing Nature of Two-Party Secure Computation. In *Advances in Cryptology: Crypto ’99*, 1999.
- [5] D. Chaum. Blind Signatures for Untraceable Payments. In *Advances in Cryptology: Crypto ’82*, 1983.
- [6] J. Camenisch, U. Maurer, and M. Stadler. Digital Payment Systems with Passive Anonymity-Revoking Trustees. In *Lecture Notes in Computer Science vol. 1146*, 1996.
- [7] J. Camenisch, J. Piveteau, and M. Stadler. Fair Blind Signatures. In *Proceedings of EuroCrypt ’95*, 1995.
- [8] S. D’Amigo and G. Di Crescenzo. Methodology for Digital Money based on General Cryptographic Tools. In *Advances in Cryptology: Eurocrypt ’94*, 1994.
- [9] G. Davida, Y. Frankel, Y. Tsionnis, and M. Yung. Anonymity Control in Electronic Cash Systems. In *Proceedings of 1st Financial Crypto*, 1997.
- [10] T. Eng and T. Okamoto. Single-Term Divisible Coins In *Advances in Cryptology: Eurocrypt ’94*, 1994.
- [11] E. Fujisaki and T. Okamoto. Practical Escrow Cash System. In *Lecture Notes in Computer Science vol. 1189*, 1997.
- [12] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology: Crypto ’86*, 1986.
- [13] O. Goldreich, L. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, 1989.
- [14] S. Goldwasser, M. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. In *SIAM Journal on Computing* 18, pp. 186-208, 1989. Preliminary version in *Proceedings of STOC ’85*, 1985.

- [15] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. In *SIAM Journal on Computing* 17(2), pp. 21-25, 1988.
- [16] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity, or all languages in NP have zero-knowledge proof systems. In *Journal of the ACM*, 38(3), pp. 691-729, 1991.
- [17] Louis Claude Guillou and Jean-Jacques Quisquater. A “paradoxical” identity-based signature scheme resulting from zero-knowledge. In *Advances in Cryptology: Crypto '88*, 1988.
- [18] M. Jakobsson and J. Muller. Improved Magic Ink Signatures Using Hints. In *Proceedings of Financial Crypto '99*, 1999.
- [19] M. Jakobsson and M. Yung. Revokable and Versatile Electronic Money. In *3rd ACM Conference on Computer and Communications Security*, 1996.
- [20] E. Mohammed, A-E. Emarah, and K. El-Shennaway. A Blind Signature Scheme Based on ElGamal Signature. In *Proceedings of the Seventeenth National Radio Science Conference, 17th NRSC '2000*, 2000.
- [21] Silvio Micali. A secure and efficient digital signature algorithm. Technical Report MIT/LCS/TM-501, Massachusetts Institute of Technology, Cambridge, MA, March 1994.
- [22] R. Molender, D. Mussington, and P. Wilson. Cyberpayments and Money Laundering: Problems and Promise. Document MR-965-OSTP/FinCEN, 1998. Available at <http://www.rand.org/publications/MR/MR965/MR965.pdf>
- [23] R. Merkle. Protocols for Public Key Cryptosystems. In *Proceedings of the 1980 Symposium on Security and Privacy*, 1980.
- [24] D. M'Raihi. Cost-Effective Payment Schemes with Privacy Regulation. In *Proceedings of ASIACRYPT '96*, 1996.
- [25] D. Naccache and S. von Solms. On Blind Signatures and Perfect Crimes. In *Computation and Security*, 1992.
- [26] T. Okamoto. An Efficient Divisible Electronic Cash Scheme. In *Advances in Cryptology: Crypto '95*, 1995.
- [27] K. Ohta and T. Okamoto. Universal Electronic Cash. In *Advances in Cryptology: Crypto '91*, 1992.
- [28] J. C. Pailles. New Protocols for Electronic Money In *Proceedings of Auscrypt '92*, 1993.
- [29] H. Peterson and G. Poupad. Efficient Scalable Fair Cash with Offline Extortion Protection. In *Lecture Notes in Computer Science vol. 1334*, 1997.
- [30] C. P. Schnorr. Efficient Identification and Signatures for Smart Cards. In *Advances in Cryptology: EUROCRYPT 89*, 1989.
- [31] T. Sander and A. Ta-Shma. Auditable, Anonymous Electronic Cash Extended Abstract In *Advances in Cryptology: Crypto '99*, 1999.