Endorsed E-Cash

Jan Camenisch
IBM Zurich
jca@zurich.ibm.com

Anna Lysyanskaya
Brown University
anna@cs.brown.edu

Mira Meyerovich
Brown University
mira@cs.brown.edu

Abstract

An electronic cash (e-cash) scheme lets a user withdraw money from a bank and then spend it anonymously. E-cash can be used only if it can be securely and fairly exchanged for electronic goods or services. In this paper, we introduce and realize endorsed e-cash. An endorsed e-coin consists of a lightweight endorsement x and the rest of the coin which is meaningless without x. We reduce the problem of exchanging e-cash to that of exchanging endorsements. We demonstrate the usefulness of endorsed e-cash by exhibiting simple and efficient solutions to two important problems: (1) optimistic and unlinkable fair exchange of e-cash for digital goods and services; and (2) onion routing with incentives and accountability for the routers. Finally, we show how to represent a set of n endorsements using just one endorsement; this means that the complexity of the fair exchange protocol for n coins is the same as for one coin, making e-cash all the more scalable and suitable for applications. Our fair exchange of multiple e-coins protocol can be applied to fair exchanges of (almost) any secrets.

Keywords E-cash, digital signatures, fair exchange, threshold cryptography

1 Introduction

The main idea of anonymous electronic cash (referred to as e-cash in the sequel), invented by David Chaum [Cha83, Cha84], is that, even though the same bank is responsible for giving out electronic coins, and for later accepting them for deposit, it is impossible for the bank to identify when a particular coin was spent (unless a user tries to spend the same coin more than once, in which case we want to catch this behavior). E-Cash has been studied extensively [CFN90, FY92, CP93, Bra93a, Bra93b, CPS94, Bra93c, SPC95, Jak95, FTY96, Tsi97, BP02].

In the past few years, there has been an explosion of e-cash research. Most work has focused on efficient withdrawal, spend, and fraud detection protocols. Camenisch et al. [CHL05] introduce compact e-cash, which allows the user to withdraw a wallet of n e-coins performing only $O(1)$ multi-base exponentiations. All n coins can be stored using a constant amount of memory; to spend a single coin requires $O(1)$ multi-base exponentiations. Wei [Wei05] shows how to efficiently trace all coins of dishonest users. Camenisch et al. [CHL06] extend compact e-cash to allow money laundering detection. Teranish and Sako [TS04], Nguyen and Safavi-Naini [NSN05], and Camenisch et al. [CHK+06] show how to use variations of compact e-cash schemes for anonymous authentication.

This paper adapts e-cash to make it useful for practical applications. It is crucial for users to have the ability to exchange e-cash for digital goods and services in a secure and *fair* fashion. A merchant should get paid only if the user gets the merchandise. However, an e-coin is really a (blind) digital signature and does not necessarily lend itself to such protocols. Prior approaches fail to provide fairness to the user: if the exchange aborts, then the user loses his privacy and, sometimes, even his money. In this paper, we introduce the idea of *endorsed* electronic cash. We let the user publish an unlimited number of *promises* of a coin. Promises of the same coin cannot be linked to each other. Each promise comes with a unique endorsement. The coin is not spent until the user gives a merchant the endorsement that goes with the promised coin. Exchanging e-cash is reduced to exchanging lightweight endorsements.

The user withdraws a *wallet coin* from the bank. In regular e-cash, the user transforms the wallet coin into an e-coin (*coin*) and gives it to a merchant. He cannot spend the same wallet coin twice. In endorsed e-cash, a user can transform a wallet coin into an unlimited number of endorsable e-coins $(\phi, x, y, coin')$. The value $coin'$ is a blinded version of *coin* and $\phi(x) = y$, where ϕ is a one-way homomorphic function. The tuple $(\phi, x, y, coin')$ should have enough information to reconstruct *coin*. The user gives the merchant an *unendorsed* coin $(\phi, y, coin')$ and saves the *endorsement*

2007 IEEE Symposium on Security and Privacy(SP'07)
0-7695-2848-1/07 $20.00 © 2007 IEEE

x for himself. The merchant must learn x in order to get the coin. This can be done via a fair exchange, or some other protocol. Thus, we can focus on designing protocols to let the merchant obtain a lightweight endorsement, rather than the entire e-coin. If an exchange fails, the user can make more unendorsed coins from the original wallet coin. None of these unendorsed coins can be linked to each other, even if one of them is eventually endorsed. We formally define endorsed e-cash in Section 2.3, and realize it using Camenisch et al. [CHL05] compact e-cash as a starting point in Section 3.2. We give an on-line variant in Section 3.4.

FAIR EXCHANGE APPLICATIONS. Suppose Alice wishes to purchase some on-line goods from Bob. Alice wants to make sure that she doesn't give away her money unless she actually gets the goods. Bob wants to make sure that he doesn't give away the goods without getting paid. This is a well-known problem called *fair exchange* [CTS95, Mic97, ASW97, ASW00]. In optimistic fair exchange [Mic97, ASW00], fairness is ensured by the existence of a trusted third party (TTP) who intervenes only if one of the players claims that something went wrong.

Prior work on fair exchange focused on exchanging digital signatures, or on exchanging a digital signature for digital goods or services. There have been several prior attempts to realize fair exchange of e-cash. Jakobsson's [Jak95] and Reiter, Wang and Wright's [RWW05] schemes' are not fair to the user: the user is not allowed to reuse a coin from a failed exchange. Asokan, Shoup and Waidner [ASW00] show how to exchange Brands' e-cash [Bra93b], but their protocol is not quite fair to the user either: if an exchange fails, a user may reuse the coin, but he cannot do so unlinkably. This weakness in all three schemes cannot be solved by a TTP. At early stages of the fair exchange protocol, the merchant can ask the TTP to terminate the exchange; however, the user would have already revealed too much information about his coin. This is a serious problem because on-line multi-party protocols often fail due to network delay, computers crashing, and etc.

In endorsed e-cash, an honest user can issue an *unlimited* number of *unlinkable* unendorsed coins corresponding to the same wallet coin and then choose which one of the coins to endorse.[1] If a fair exchange fails, the user can throw out the corresponding unendorsed coin, and use the same wallet coin to generate another one.

[1] The user can make $O(2^k)$ promises of the same e-coin. To create an e-coin, the user makes some random choices; if the user makes the same choices twice, this results in two identical e-coins that can be linked to each other. The endorsement for one is an endorsement to the other. In this case, the user risks not only losing some privacy, but even being accused of double spending. For an honest user, this scenario can happen with only negligible probablity.

We achieve this by applying Asokan, Shoup, and Waidner's [ASW00] optimistic fair exchange of pre-images of a homomorphic one-way function protocol to *endorsements*, in Section 4.1.

In the real world, it is often impossible to buy digital goods with a single coin. The obvious solution is to run a fair exchange of all the coins together: to do this, a user must verifiably escrow all n endorsements. If the TTP gets involved, it has to store, and later decrypt, all n escrows. (A verifiable escrow costs about ten times more than, say, an ElGamal encryption.) Surprisingly, it turns out that it is possible to compress n endorsements into one! The burden on the TTP is now the same regardless of how much the digital goods cost. Details in Section 4.2.

BUYING SERVICES. E-cash can be used to purchase digital services. Suppose a user asks a service-provider to perform several tasks on its behalf, for example, to negotiate with various certification authorities or to engage in a series of financial transactions. The user does not want to pay the service-provider unless *all* of the tasks are completed, because often, performing some of the tasks is no better or even worse than performing none of them. To solve this problem, we introduce threshold endorsed e-cash (Section 4.3) where the user can create n endorsements for one e-coin, of which any m suffice to reconstruct the coin.

For a concrete example, consider anonymous remailers and onion routing schemes, such as [Cha81, DDM03, CL05]. A user sends an encrypted message via a chain of routers; each router peels off a layer of encryption before passing the message on to the next router. The user needs to give routers an incentive to forward the message. If the user simply includes an e-coin in each layer of encryption, then a router gets paid even if it does not forward the message. Reiter et al. [RWW05] suggest including a "ripped" e-coin [Jak95] in each layer, along with a verifiable encryption of the second half of the coin under the next router's public-key. A router would pass the message and hope that the next router in the chain is honest and would send the second half of the coin back. However, the next router has no incentive to do so – because it does not need anything from the previous router. Even worse, the user loses the coin regardless of whether a router passes its message.

Threshold endorsed e-cash easily overcomes this problem. We set $n = m = 2$; each layer of the onion reveals a threshold unendorsed e-coin. To get the endorsement, a router must contact both the previous and next routers in the chain. Both are motivated to talk to him because he has endorsements for their coins. A dishonest router does not get paid and the user does not lose the e-coin (as long as we can enforce a timeout after which a dishonest router cannot suddenly contact

2007 IEEE Symposium on Security and Privacy(SP'07)
0-7695-2848-1/07 $20.00 © 2007 IEEE

other routers on the chain, complete the exchange, and deposit the e-coin).

PRACTICAL APPLICATIONS. Our scheme is efficient enough to be deployed on most computing devices, from PCs to smartcards. We believe that e-cash is going to become more important in today's electronic world. In large peer-to-peer systems, participants have to trust others to perform services for them. Anonymous remailers and onion routing schemes [Cha81, DDM03, CL05] are a good example: to provide anonymity for one peer, several peers have to be on-line and available to serve as routers. Other examples are peer-to-peer systems for publishing [Coh03] and backing up [McC01] data. Participants in a peer-to-peer system perform services to earn brownie points and subsequently use them to buy services from others. Peer-to-peer systems already have economies of their own [ADS03] and for the sake of privacy, these economies should utilize e-cash.

ORGANIZATION. We introduce our notation, state our complexity assumptions and define security for compact and endorsed e-cash in Section 2. We construct off-line and on-line endorsed e-cash in Sections 3.2 and 3.4, respectively. Finally, Section 4 contains endorsed e-cash protocols: fair exchange for a single e-coin in Section 4.1, efficient fair exchange of multiple e-coins in Section 4.2, and threshold endorsed e-cash in Section 4.3.

2 Notation and Definitions

We say ν is a negligible function in k if, for all c, and all large enough k, $\nu(k) < n^c$. A *homomorphic* function $f : D \to R$ has the property that $f(a \oplus b) = f(a) \otimes f(b)$, where $\oplus$ and $\otimes$ are the group operations in D and R. If f is a *one-way* function, then given $f(x)$ (for some randomly chosen x), any polynomial time algorithm has a negligible chance of guessing an x' such that $f(x') = f(x)$. Let G be a group of prime order where the discrete logarithm problem is hard and $g_1, \ldots, g_{k+1}$ be generators of G. The Pedersen commitment [Ped92] of $(x_1, \ldots, x_k)$ with randomization factor x_{k+1} is: $\mathsf{Ped}(x_1, \ldots, x_k; x_{k+1}) = \prod_{1 \le i \le k+1} g_i^{x_i}$.

2.1 Security Assumptions

We list the security assumptions used by our endorsed e-cash protocol.

Definition 2.1 (Discrete Logarithm Assumption)**.** *Let G be a group of prime order. Suppose we randomly choose g, a generator of G, and $x \in \mathbb{Z}_q$. The Discrete Logarithm Assumption states that any PPTM that gets as input $y = g^x$ can compute x with, at most, negligible probability.*

Definition 2.2 (Diffie-Hellman Assumption)**.** *Let G be a group of prime order. Suppose we randomly choose g, a generator of G, and $x, y \in \mathbb{Z}_q$. The Diffie-Hellman Assumption states that any PPTM that gets as input (g, g^x, g^y) can compute g^{xy} with, at most, negligible probability.*

Definition 2.3 (q-DHI Assumption ([MSK02]))**.** *Let G be a group of prime order. Suppose we randomly choose g, a generator of G, and $x \in \mathbb{Z}_q$. The q-Diffie-Hellman Inversion assumption states that no PPTM can compute $g^{1/x}$ given $(g, g^x, \ldots, g^{(x^q)})$.*

Definition 2.4 (q-DBDHI Assumption ([BB04]))**.** *Let $G, \hat{G}$ be groups of prime order and let $e : G \times G \to \hat{G}$ be a bilinear map. Choose a random g, a generator of G, and $x \in \mathbb{Z}_q$. The q-Decisional Diffie-Hellman Inversion assumption states that no PPTM can distinguish $e(g, g)^{1/x}$ from random, even after seeing $(g, g^x, \ldots, g^{(x^q)})$.*

Definition 2.5 (Strong RSA Assumption ([BP97]))**.** *Let n be an RSA modulus. Suppose we choose a random $z \in \mathbb{Z}_n^*$. The Strong RSA Assumption states that a PPTM on input (n, z) can output values $y \in \mathbb{Z}_n^*$ and $r > 1$ such $y^r = z \bmod n$ with at most negligible probability.*

Definition 2.6 (Paillier Assumption ([Pai99]))**.** *Let n be an RSA modulus and $P = \{a^n | a \in \mathbb{Z}_{n^2}\}$. The Paillier assumption states that no PPTM can distinguish a random element of P from a random element of $\mathbb{Z}_{n^2}$.*

2.2 Definition of E-Cash

Suppose we have an e-cash system $\mathcal{EC}$ = ($\mathsf{BKeygen}$, $\mathsf{UKeygen}$, $\mathsf{Withdraw}$, Spend, $\mathsf{Deposit}$, $\mathsf{PublicSecurityProtocols}$). (Different e-cash systems have different sets of associated algorithms; for consistency, we will use the Camenisch et al. definition [CHL05].) We briefly overview each of the protocols and their security properties. We will give rigorous treatment to only those protocols whose definition and security properties are relevant to understanding endorsed e-cash.

We have three types of players: banks, users and merchants. Merchants are a subset of users. We generally use $\mathcal{B}$ to denote a bank, $\mathcal{M}$ to denote a merchant and $\mathcal{U}$ to denote a user. When we write $\mathsf{Protocol}(\mathcal{U}(x), \mathcal{B}(y))$ we mean that there is a protocol called $\mathsf{Protocol}$ between a user $\mathcal{U}$ and a bank $\mathcal{B}$ in which the private input of $\mathcal{U}$ is x and the private input of $\mathcal{B}$ is y.

A user can withdraw a wallet W of n coins from his bank account. An e-cash system defines a set of protocols for transferring wallets and coins between players

2007 IEEE Symposium on Security and Privacy(SP'07)
0-7695-2848-1/07 $20.00 © 2007 IEEE

and for handling cheaters. A protocol can either be a function invoked by a single player, in which case we list the arguments to the function, or an interactive two-party protocol, in which case we list the relevant parties and the private inputs each one uses.

$\mathsf{BKeygen}(1^k, \mathit{params})$ A bank $\mathcal{B}$ invokes BKeygen to generate $(pk_{\mathcal{B}}, sk_{\mathcal{B}})$, its public/private-key pair.

$\mathsf{UKeygen}(1^k, \mathit{params})$ A user $\mathcal{U}$ (or a merchant $\mathcal{M}$) invokes UKeygen to generate $(pk_{\mathcal{U}}, sk_{\mathcal{U}})$, its public/private-key pair.

$\mathsf{Withdraw}(\mathcal{U}(pk_{\mathcal{B}}, sk_{\mathcal{U}}, n), \mathcal{B}(pk_{\mathcal{U}}, sk_{\mathcal{B}}, n))$ This is a protocol between a user $\mathcal{U}$ and a bank $\mathcal{B}$ that lets the user withdraw n coins from his bank account. The user gets either a wallet W of n coins, or an error message. The bank gets either some trace information that it stores in a database, or an error message.

$\mathsf{Spend}(\mathcal{U}(W, pk_{\mathcal{M}}), \mathcal{M}(sk_{\mathcal{M}}, pk_{\mathcal{B}}, n))$ This is a protocol between a user $\mathcal{U}$ and a merchant $\mathcal{M}$ that transfers one coin from the user's wallet W to the merchant. The merchant gets an e-coin *coin* and the user updates his wallet to contain one less coin.

$\mathsf{Deposit}(\mathcal{M}(sk_{\mathcal{M}}, \mathit{coin}, pk_{\mathcal{B}}), \mathcal{B}(pk_{\mathcal{M}}, sk_{\mathcal{B}}))$ This is a protocol between a merchant $\mathcal{M}$ and a bank $\mathcal{B}$ that lets the merchant deposit a coin he got from a customer into his bank account.

$\mathsf{PublicSecurityProtocols}(\mathit{protocol}, \mathit{params}, \mathit{arglist})$ This is a set of functions that can be invoked by anybody to identify double spenders and verify their guilt. The bank *finds* double-spenders, but it must be able to *convince* everyone else. The Camenisch et al. protocols [CHL05] include $\mathsf{Identify}(\mathit{params}, \mathit{coin1}, \mathit{coin2})$ to identify a double spender, $\mathsf{VerifyGuilt}(\mathit{params}, \mathit{coin}, pk_{\mathcal{U}}, \mathit{proof})$ to publicly verify that user $\mathcal{U}$ had double spent a coin, $\mathsf{Trace}(\mathit{params}, \mathit{coin}, pk_{\mathcal{U}}, \mathit{proof}, \mathit{database})$ to find all coins spent by a guilty user, VerifyOwnership $(\mathit{params}, \mathit{coin}, \mathit{proof}, pk_{\mathcal{U}})$ to verify that a guilty user spent a particular coin. The exact set of functions depends on the e-cash system and its desired security properties.

The security properties of an e-cash system depend on the model we use: plain, random oracle, common random string, etc. Here we sketch what an adversary must do to defeat the e-cash system and explain where the properties of the security model come into play; we refer the reader to Camenisch et al. [CHL05]. We require four properties from an e-cash system:

Correctness: If an honest user runs Withdraw with an honest bank, then neither outputs error; if an honest user runs Spend with an honest merchant, then the merchant accepts the coin.

Anonymity: Even if a malicious bank conspires with one or more malicious merchants, the bank cannot link a user to any coins he spends. We create a simulator $\mathcal{S}$ and give it special powers (e.g. control of random oracle, ability to generate common parameters, control of key generation). The simulator should be able to run the Spend protocol without knowing any information about any user's wallet or public/secret-key pair.

Formally, we create an adversary $\mathcal{A}$ that plays the part of the bank and of all merchants. $\mathcal{A}$ creates the bank's public-key $pk_{\mathcal{B}}$. Then, $\mathcal{A}$ gets access to an interface Game that plays either the real or ideal game; $\mathcal{A}$ must determine which. $\mathcal{A}$ can make four types of queries to Game:

$\mathsf{GameSetup}(1^k)$ generates the public parameters *params* and private parameters *auxsim* for $\mathcal{S}$.

$\mathsf{GameGetPK}(i)$ returns the public-key of user $\mathcal{U}_i$, generated by $\mathsf{UKeygen}(1^k, \mathit{params})$.

$\mathsf{GameWithdraw}(i)$ runs the Withdraw protocol with user $\mathcal{U}_i$: $\mathsf{Withdraw}(\mathcal{U}_i(pk_{\mathcal{B}}, sk_i, n), \mathcal{A}(\mathit{state}, n))$. (We use *state* to denote the state of the adversary; it is updated throughout the course of protocol execution). We call W_j the wallet generated the jth time protocol Withdraw is run.

$\mathsf{GameSpend}(j)$ in the real game, this runs the spend protocol with the user $\mathcal{U}$ that holds the wallet W_j: $\mathsf{Spend}(\mathcal{U}(W_j), \mathcal{A}(\mathit{state}, n))$. In the ideal game, $\mathcal{S}$ pretends to be the user: $\mathsf{Spend}(\mathcal{S}(\mathit{params}, \mathit{auxsim}, pk_{\mathcal{B}}), \mathcal{A}(\mathit{state}, n))$; $\mathcal{S}$ does not have access to the wallet W_j or know who owns it.

An adversary is *legal* if it never asks a user to double-spend a coin: for all j, the adversary never calls $\mathsf{GameSpend}(j)$ more than n times (where n is the size of the wallet). An e-cash scheme *preserves anonymity* if, for all $pk_{\mathcal{B}}$, no computationally bounded *legal* adversary can distinguish between the real game and the ideal game with more than negligible probability.

Balance: No group of dishonest users and merchants should be able to deposit more coins than they withdraw. We assume that each coin has a serial number (generated during the Withdraw protocol) We create a knowledge extractor $\mathcal{X}$ that executes the Withdraw protocol with u dishonest users and generates un coin serial numbers: $S_1, \ldots, S_{un}$ (we assume each user withdraws n coins). No adversary should be able to successfully deposit a coin with serial number S unless $S \in \{S_1, \ldots, S_{un}\}$. Again, $\mathcal{X}$ must have additional powers, such as control of the random oracle or special knowledge about public parameters.

2007 IEEE Symposium on Security and Privacy(SP'07)
0-7695-2848-1/07 $20.00 © 2007 IEEE

Culpability and Exculpability: Any user that runs Spend twice on the same coin should be caught by the bank; however, a malicious bank should not be able to conspire with malicious merchants to frame an honest user for double-spending. We omit the specifics of these definitions and refer the reader to Camenisch et al. [CHL05].

2.3 Definition of Endorsed E-Cash

Endorsed e-cash is similar to E-cash. The only difference is that spending a coin is split into two stages. In the first stage, a user gives a merchant a blinded version of the coin, a.k.a. an *unendorsed coin*. An unendorsed coin is not a real coin and cannot be deposited with the bank. A user is allowed to issue unendorsed coins as often as he wants — it should be impossible to link two unendorsed versions of the same coin. (This is the chief difference between our solution and that of Jakobsson [Jak95] and Asokan et al. [ASW00]). A user can endorse a coin by giving a particular merchant the information he needs to transform the unendorsed coin into a real coin (i.e. an *endorsed coin*) that can be deposited with the bank. As long as a user endorses at most one version of the same wallet coin, he is not a double-spender and cannot be identified.

An endorsed e-cash system is almost identical to a regular e-cash system, except Spend is replaced by SplitCoin, ESpend, and Reconstruct. We define the three new protocols:

SplitCoin($params, W_j, pk_{\mathcal{B}}$) A user $\mathcal{U}$ can take a coin from his wallet and generate $(\phi, x, y, coin')$. The value $coin'$ is a blinded version of the e-coin. The function ϕ is a one-way homomorphic function, such that $\phi(x) = y$. The tuple $(\phi, x, y, coin')$ should have enough information to reconstruct the e-coin.

ESpend($\mathcal{U}(W, pk_{\mathcal{M}}), \mathcal{M}(sk_{\mathcal{M}}, pk_{\mathcal{B}}, n)$) This is the endorsed spend protocol. The user $\mathcal{U}$ privately runs SplitCoin to generate $(\phi, x, y, coin')$. The user gives the merchant $(\phi, y, coin')$, but keeps x for himself. The merchant uses $coin'$ to verify the validity of the unendorsed coin.

Reconstruct($\phi, x, y, coin'$) This function (typically used by a merchant) reconstructs a coin that can be deposited with the bank *if and only if* $\phi(x) = y$.

An endorsed e-cash scheme should have the same properties of correctness, anonymity, balance, culpability and exculpability as an e-cash scheme. However, the definitions must be slightly modified to fit the new set of protocols:

Correctness: (Informally), if an honest user runs Withdraw with an honest bank, then neither will output an error message; if an honest user runs SplitCoin and gives the resulting $(\phi, y, coin')$ to an honest merchant via the ESpend protocol, the merchant will accept; if an honest merchant gets $(\phi, y, coin')$ from an honest user and learns the value $x = \phi^{-1}(y)$, then he'll be able to use Reconstruct to generate a valid coin that an honest bank will accept during the Deposit protocol.

Anonymity: Splitting a coin into two pieces: $(\phi, y, coin')$ and x should not increase the ability of a consortium of a malicious bank and merchants to link a coin to a user. Nor should an adversary be able to link two unendorsed versions of the same coin to each other. Once again, we create a simulator $\mathcal{S}$ and give it special powers. The simulator should be able to run the ESpend protocol without knowing any information about any user's wallet or public/secret-key pair.

Formally, we create an adversary $\mathcal{A}$ that plays the part of the bank and of all merchants. $\mathcal{A}$ creates the bank's public-key $pk_{\mathcal{B}}$. Then, $\mathcal{A}$ gets access to an interface Game that plays either a real game or an ideal game; $\mathcal{A}$ must determine which. $\mathcal{A}$ can make five types of queries to Game:

GameSetup(1^k) generates the public parameters $params$ and private parameters $auxsim$ for $\mathcal{S}$.

GameGetPK(i) returns the public-key of user $\mathcal{U}_i$, generated by UKeygen($1^k, params$).

GameWithdraw(i) runs the Withdraw protocol with user $\mathcal{U}_i$: Withdraw($\mathcal{U}_i(pk_{\mathcal{B}}, sk_i, n), \mathcal{A}(state, n)$). We call W_j the wallet generated the jth time the protocol Withdraw is run.

GameESpend(j, J) gives the adversary an unendorsed coin number J from wallet W_j. In the real game, GameESpend runs the ESpend protocol with the user $\mathcal{U}$ that holds the wallet W_j: ESpend($\mathcal{U}(W_j, J, pk_{\mathcal{B}}), \mathcal{A}(state, n)$). In the ideal game, $\mathcal{S}$ plays the part of the user and runs the protocol: ESpend($\mathcal{S}(params, auxsim, pk_{\mathcal{B}}), \mathcal{A}(state, n)$). $\mathcal{S}$ knows nothing about the wallet W_j, the particular coin J requested, or the user who owns it. In the end, the adversary gets the unendorsed coin $(\phi, y, coin')$.

GameEndorse($\phi, y, coin'$) returns either the endorsement $x = \phi^{-1}(y)$ or an error message if the protocol GameESpend has not previously issued $(\phi, y, coin')$.

An adversary is called *legal* if it never asks a user to double-spend. Suppose two separate calls to GameESpend(j, J) result in the responses $(\phi, y_1, coin'_1)$ and $(\phi, y_2, coin'_2)$. A legal adversary never calls both GameEndorse($\phi, y_1, coin'_1$) and

2007 IEEE Symposium on Security and Privacy(SP'07)
0-7695-2848-1/07 $20.00 © 2007 IEEE

$\mathsf{GameEndorse}(\phi, y_2, coin'_2)$. An endorsed e-cash scheme preserves anonymity if no computationally bounded *legal* adversary can distinguish between the real and ideal game with more than negligible probability.

Balance: The balance property remains the same.

Culpability and Exculpability: We combine SplitCoin, ESpend, and Reconstruct to create a protocol SPEND that corresponds to the Spend protocol of a standard e-cash scheme. We need to show that the e-cash system $\mathcal{EC}$ = (BKeygen, UKeygen, Withdraw, SPEND, Deposit, PublicSecurityProtocols) meets the culpability and exculpability guarantees of a standard e-cash system. We define $\mathsf{SPEND}(\mathcal{U}(W, pk_{\mathcal{M}}), \mathcal{M}(sk_{\mathcal{M}}, pk_{\mathcal{B}}, n))$ as follows: First, $\mathcal{U}$ calls $\mathsf{SplitCoin}(params, W_j, pk_{\mathcal{B}})$ to generate the tuple $(\phi, x, y, coin')$ and sends it to $\mathcal{M}$. When $\mathcal{M}$ receives $(\phi, x, y, coin')$, he verifies that $(\phi, y, coin')$ is valid (as in ESpend), and checks if $\phi(x) \neq y$ coin. If either test fails, $\mathcal{M}$ rejects. Otherwise, $\mathcal{M}$ creates the corresponding endorsed coin $coin = \mathsf{Reconstruct}(\phi, x, y, coin')$. $\mathcal{M}$ stores $coin$ until he is ready to deposit it.

The culpability and exculpability properties provide protection if the user issues only one unendorsed coin per wallet coin – in this case, endorsed e-cash reduces to standard e-cash. So what prevents dishonest merchants from using an endorsement from one coin to generate endorsements for other coins? If a merchant successfully deposits a falsely endorsed coin with the bank, then he violates the balance property. If the merchant uses the fake endorsement to frame a user for double-spending, then he violates anonymity.

3 Endorsed E-Cash Instantiation

In this section we describe how to build an endorsed e-cash system from the Camenisch, Hohenberger and Lysyanskaya ([CHL05] Section 4.1) e-cash system, referred to as CHL in sequel. All we have to do is split the CHL Spend protocol into (SplitCoin, Reconstruct, ESpend). We review the CHL Spend protocol in Section 3.1. Then we modify it to create an endorsed e-cash system in Section 3.2 and prove it is secure in Section 3.3. We construct a CHL-like on-line endorsed e-cash system in Section 3.4.

3.1 CHL Compact E-Cash

CHL compact e-cash lets users withdraw several coins at once. A user has a secret-key $u \in \mathbb{Z}_q$ and public-key g^u. To withdraw n coins, the user randomly chooses $s, t \in \mathbb{Z}_q$ and obtains from the bank a CL-signature σ on (u, s, t). A CL-signature [CL02, CL04] lets the bank sign a message without learning what it is (though the bank learns some information about σ). Now the user has a wallet of n coins: $(0, u, s, t, \sigma), \ldots, (n-1, u, s, t, \sigma)$.

To pay a merchant, the user constructs an e-coin (S, T, Φ, R) from the wallet coin (J, u, s, t, σ) (see Algorithm 3.1). S is a unique (with high probability) serial number, (T, R) are needed to trace double-spenders — knowing two different (T, R) values corresponding to the same wallet coin lets the bank learn the user's identity, Φ is a zero-knowledge proof that tells the merchant and bank that the e-coin is valid, and R is as hash of the contract between the user and merchant and should be unique to every transaction (this lets the bank use (T, R) to catch double-spenders).

To deposit and e-coin, the merchant gives (S, T, Φ, R) to the bank, along with his public-key. The bank checks whether it has already seen a coin with serial number S – if yes, then the bank knows that *somebody* is trying to double-spend because S is supposed to be unique. If it has seen (S, R) before, then the merchant is at fault because R is unique to every transaction If the bank hasn't seen (S, R) before, then the user is at fault and the bank uses the values $(S, T_{old}, \Phi_{old}, R_{old})$ and (S, T, Φ, R) to learn the double-spending user's identity. CHL finds double-spenders in a manner similar to Chaum et al. [CFN90], but it only learns the user's public-key, and *not* his secret-key (Camenisch et al's extended solution also reveals the secret-key). This distinction has great significance to fair exchange (Section 4).

Global parameters: Let k be the security parameter. All computation is done in a group G, of prime order $q = \Theta(2^k)$, with generator g. We assume there is a public-key infrastructure.

Spend lets a user $\mathcal{U}$ pay a merchant $\mathcal{M}$ the wallet coin (J, u, s, t, σ): First, the user and merchant agree on a contract *contract* (we assume each contract is unique per merchant). The merchant gives the user his public key $pk_{\mathcal{M}}$. Then, the user runs CalcCoin, as defined in Algorithm 3.1, to create the coin (S, T, Φ, R) and sends it to the merchant. Finally, the merchant verifies Φ to check the validity of the coin (S, T, Φ, R).

Efficiency: CalcCoin uses the Dodis-Yampolskiy pseudo-random function [DY05] to instantiate $F_s(x) = g^{1/(s+x+1)}$. As a result, a user must compute seven multi-base exponentiations to build the commitments and eleven more for the proof. The merchant and bank need to do eleven multi-base exponentiations to check that the coin is valid.

Security: CHL requires (1) the security of a CL-signature, which depends on the Strong RSA Assumption, (2) the zero-knowledge proof (or argument) system, which relies on the Strong RSA Assumption and the Random Oracle Model, (3) the collision-resistant

Algorithm 3.1: CalcCoin

Input: $pk_{\mathcal{M}} \in \{0,1\}^*$ merchant's public key,
$contract \in \{0,1\}^*$
User Data: u private key, g^u public key,
(s,t,σ,J) a wallet coin
$R \leftarrow H(pk_{\mathcal{M}}||info)$;
$S \leftarrow F_s(J)$;
$T \leftarrow g^u F_t(J)^R$;
Calculate ZKPOK Φ of (J,u,s,t,σ) such that:
$0 \leq J < n$
$S = F_s(J)$
$T = g^u F_t(J)^R$
$\mathrm{VerifySig}(pk_{\mathcal{B}},(u,s,t),\sigma) = true$
F is a pseudo-random function, H is a collision-resistant hash function.
return (S,T,Φ,R)

hash function H and (4) the security of the pseudo-random function $F_s(x)$, which if instantiated as the Dodis-Yampolskiy pseudo-random function, depends on the q-DHI and q-DBDHI assumptions

3.2 Endorsed E-Cash Construction

Our endorsed e-cash construction is based on CHL. The wallet coin (J,u,s,t,σ) is the same as before, but the unendorsed coin is a blinded version of the CHL e-coin. Instead of giving the merchant (S,T,Φ,R), the user chooses a random endorsement (x_1,x_2,x_3) and calculates (S',T',Φ',R,y), where $S' = Sg^{x_1}$, $T' = Tg^{x_2}$ and $y = \mathsf{Ped}(x_1,x_2;x_3)$. The value Φ' is a zero-knowledge proof that the unendorsed coin is valid. Once the merchant learns the endorsement, he can easily reconstruct (S,T,Φ',R), which along with y and (x_1,x_2,x_3) constitutes an endorsed coin that can be deposited with the bank. The user can generate as many unendorsed versions of the same wallet coin as he wants by choosing different endorsements. However, if he endorses two versions of the same wallet coin, the bank will identify him using the same method as in CHL.

Global parameters: Same as in CHL. Additionally, let g,h_1,h_2 be elements in G whose discrete logarithms with respect to each other are unknown. We define the homomorphic one-way function $\phi : \mathbb{Z}_q^3 \rightarrow G$, where $\phi(a,b,c) = h_1^a h_2^b g^c$. We split the public parameters $params = (params_{CHL}, params_{ZK})$, where $params_{ZK}$ is used for the ZKPOK in the $\mathsf{SplitCoin}$ protocol and $params_{CHL}$ is used for everything else (and is, in fact, the same as in the CHL system).

$\mathsf{SplitCoin}$, defined in Algorithm 3.2, creates an endorsable coin $(S',T',\Phi',R,(x_1,x_2,x_3),y)$, where (S',T',Φ',R,y) is the unendorsed coin and (x_1,x_2,x_3) is the endorsement (with $\phi(x_1,x_2,x_3) = y$). The values S' and T' are blinded versions of S and T and Φ' is the zero-knowledge proof that S' and T' are formed correctly. The merchant verifies Φ' during the ESpend protocol.

When the merchant receives the endorsement (x_1,x_2,x_3) for his unendorsed coin (S',T',Φ',R,y) he calls $\mathsf{Reconstruct}$ to create an endorsed coin $(S = S'/g^{x_1}, T = T'/g^{x_2}, \Phi', R, (x_1,x_2,x_3), y)$. The endorsed coin is almost identical to the original coin (S,T,Φ,R), except that Φ' is a zero-knowledge proof of slightly different information. Possession of that information is sufficient to create a valid CHL coin and the bank can safely accept it. The bank can also identify double-spenders because S,T,R are constructed the same way as in the CHL Spend protocol.

Algorithm 3.2: SplitCoin

Input: $pk_{\mathcal{M}} \in \{0,1\}^*$ merchant's public key,
$contract \in \{0,1\}^*$
User Data: u private key, g^u public key,
(s,t,σ,J) a wallet coin
$R \leftarrow H(pk_{\mathcal{M}}||contract)$;
$x_1,x_2,x_3 \leftarrow \mathbb{Z}_q$;
$y \leftarrow \phi(x_1,x_2,x_3)$;
$S' \leftarrow F_s(J)g^{x_1}$;
$T' \leftarrow g^u F_t(J)^R g^{x_2}$;
Calculate ZKPOK Φ' of $(J,u,s,t,\sigma,x_1,x_2,x_3)$ such that:
$y = h_1^{x_1} h_2^{x_2} g^{x_3}$
$0 \leq J < n$
$S' = F_s(J)g^{x_1}$
$T' = g^u F_t(J)^R g^{x_2}$
$\mathrm{VerifySig}(pk_{\mathcal{B}},(u,s,t),\sigma) = true$
return $(S',T',\Phi',R,(x_1,x_2,x_3),y)$

Efficiency: $\mathsf{SplitCoin}$ is very similar to $\mathsf{CalcCoin}$; it requires two more multi-base exponentiation from the user, one to compute y and one due to its inclusion in the proof, and one more multi-base exponentiation from the merchant and bank to verify the proof. (Note: we compute T' slightly different from CHL, but this has a negligible effect on the computation.)

Security: our endorsed e-cash system requires the same assumptions as CHL.

3.3 Security

Theorem 3.1. *The endorsed e-cash system described in Section 3.2 meets the definition of a secure endorsed e-cash system.*

Proof. Correctness. It is easy to see the system is correct because the key values S,T,R are identical to the CHL e-cash system.

2007 IEEE Symposium on Security and Privacy(SP'07)
0-7695-2848-1/07 $20.00 © 2007 IEEE

Anonymity. We construct an algorithm $\mathcal{S}$ that impersonates all honest users of the endorsed e-cash system without access to their data during the ESpend protocol. (Recall, in our definition, the adversary accesses an interfaces Game, which either invokes real users or $\mathcal{S}$). $\mathcal{S}$ will use $\mathcal{S}_{CHL}$, the simulator for the CHL Spend protocol, and $\mathcal{S}_{ZK}$, the simulator for the zero-knowledge system, as building blocks. We will show that any adversary $\mathcal{A}$ that can distinguish when the interface Game plays the real game with real users or the ideal game using $\mathcal{S}$ can either (1) break the anonymity of CHL or (2) violate the zero-knowledge property of the ZKPOK system.

$\mathcal{S}$ gets as input $(params, auxsim, pk_{\mathcal{B}})$. The endorsed e-cash system generated $(params, auxsim)$ during GameSetup; some of those parameters are intended for $\mathcal{S}_{CHL}$ and $\mathcal{S}_{ZK}$: $(params_{CHL}, auxsim_{CHL})$ is intended for $\mathcal{S}_{CHL}$ and $(params_{ZK}, auxsim_{ZK})$ is for $\mathcal{S}_{ZK}$.

$\mathcal{S}$ has to simulate ESpend. It gets $(contract, pk_{\mathcal{M}})$ from $\mathcal{A}$. $\mathcal{S}$ executes $\mathsf{Spend}(\mathcal{S}_{CHL}(params_{CHL}, auxsim_{CHL}), \mathcal{S}(contract, pk_{\mathcal{M}}, pk_{\mathcal{B}}, n))$ (n is the size of the wallets), pretending to be a merchant. $\mathcal{S}$ does not need the merchant's secret-key for the Spend protocol. $\mathcal{S}_{CHL}$ gives $\mathcal{S}$ some coin (S, T, Φ, R). $\mathcal{S}$ pretends to run SplitCoin. First it randomly generate (x_1, x_2, x_3). Then it uses the the "endorsement" to calculate: $y = \phi(x_1, x_2, x_3)$, $S' = Sg^{x_1}$, and $T' = Tg^{x_2}$. Then it calls $\mathcal{S}_{ZK}(params_{ZK}, auxsim_{ZK})$ to generate a fake proof Φ'. $\mathcal{S}$ sets $coin' = (S', T', \Phi', R, y)$. It stores $(\phi, (x_1, x_2, x_3), y, coin')$ in a database for later use and returns $(\phi, y, coin')$ to the adversary.

We prove $\mathcal{S}$ is indistinguishable from real users via a hybrid argument. Consider an algorithm $\mathcal{S}_1$ that acts just like a real user, but after constructing a legitimate unendorsed coin, invokes $\mathcal{S}_{ZK}$ to create a fake proof Φ'. If $\mathcal{A}$ can distinguish $\mathcal{S}_1$ from a real user, $\mathcal{A}$ violates the zero-knowledge property of the ZKPOK system. Now consider algorithm $\mathcal{S}_2$ that generates unendorsed coins using $\mathcal{S}_{CHL}$ and $\mathcal{S}_{ZK}$, but makes sure that all unendorsed versions of the same coin have the same serial number. In this case, if $\mathcal{A}$ can distinguish $\mathcal{S}_1$ from $\mathcal{S}_2$, $\mathcal{A}$ violates the anonymity of CHL. Finally, by the definition of SplitCoin, the S' and T' are information theoretically independent of the real serial number. Therefore, $\mathcal{S}_2$ is indistinguishable from $\mathcal{S}$. By the hybrid argument, no adversary can tell when Game is playing the ideal game or the real game.

Balance. We need to show that no consortium of users and merchants can cheat an honest bank. Suppose we have an adversary $\mathcal{A}$ that can break the balance property of our endorsed e-cash system. $\mathcal{A}$ executes the Withdraw protocol u times to withdraw un coins (assuming n coins per wallet). We take the knowledge extractor $\mathcal{X}$ from the CHL system and use it to generate serial numbers $S_1, \ldots, S_{un}$ from all the invocations of Withdraw (recall that our endorsed e-cash uses the same Withdraw protocol as CHL). Eventually, $\mathcal{A}$ produces an endorsed coin $(S, T, \Phi', R, (x_1, x_2, x_3), y)$ that the bank accepts, but $S \notin S_1, \ldots, S_{un}$. Since the bank accepted the endorsed coin, this implies that $\phi(x_1, x_2, x_3) = y$ and Φ' is valid. Since Φ' is formed by a sound ZKPOK system, $\mathcal{A}$ knows values J, u, s, t, σ such that: (1) $S' = Sg^{x_1} = F_s(J)g^{x_1}$, (2) $T' = Tg^{x_2} = F_t(J)^R g^{x_2}$, and (3) $\text{VerifySig}(pk_{\mathcal{B}}, (u, s, t), \sigma) = true$.

Therefore, we can use $\mathcal{A}$ to create a proof Φ such that the CHL bank accepts the coin (S, T, Φ, R). We construct a reduction that breaks the security of the CHL scheme by playing middleman in the Withdraw and Deposit invocations that $\mathcal{A}$ makes. The reduction can set up the public parameters for the endorsed e-cash ZKPOK, and exploit them to extract the values u, s, t, σ from $\mathcal{A}$. As a result, it can construct a valid CHL ZKPOK for coins that $\mathcal{A}$ tries to deposit.

Culpability and Exculpability. Since Reconstruct creates a coin $(S, T, \Phi', R, (x_1, x_2, x_3), y)$ where (S, T, R) are the same as in the CHL system, the CHL PublicSecurityProtocols can remain unchanged. Therefore, culpability and exculpability are preserved. □

3.4 On-line Endorsed E-cash

On-line e-cash lets merchants verify with a permanently available (i.e., on-line) bank whether an e-coin was previously spent. Double-spending is detected *before* it happens.

The Spend protocol would consist of three stages. First, the user gives the merchant an e-coin serial number. Next the merchant verifies with the bank that the e-coin has not yet been spent. Finally, the user and merchant perform a fair exchange of the e-coin's endorsement and the promised good or service.

For the sake of efficiency, our on-line endorsed e-cash system makes the user give the merchant the e-coin serial number S in the clear *before* the start of the fair exchange. This lets the bank quickly check whether the e-coin has been spent. (If the user sent a blinded version of the serial number, then he and the bank would have to go through an onerous zero-knowledge proof that the promised e-coin's serial number is not in the database of spent e-coins.) Unfortunately, the user now sacrifices some anonymity because e-coins can be linked to each other, if not the user.

We now describe the on-line endorsed e-cash Spend protocol in detail.

The user sends S in the clear along with a timeout value *timeout* that tells the bank when the unendorsed coin expires. He also generates an endorsement x and calculates $y = \phi(x)$. The user creates a signature V on

2007 IEEE Symposium on Security and Privacy(SP'07)
0-7695-2848-1/07 $20.00 © 2007 IEEE

$(pk_{\mathcal{M}}, contract, y, timeout)$ using S as the verification key (details later). Since the double-spending equation T is no longer needed, the Withdraw protocol generates a shorter wallet $W = (J, u, s, \sigma)$.

Algorithm 3.3: SplitOnLineCoin

Input: $pk_{\mathcal{M}} \in \{0,1\}^*$ merchant's public key, $contract \in \{0,1\}^*$, *timeout* expiration time

User Data: u private key, g^u public key, (s, σ, J) a wallet coin

$x \leftarrow \mathbb{Z}_q$;
$y \leftarrow g^x$;
$S \leftarrow F_s(J)$;
$R \leftarrow pk_{\mathcal{M}}||contract||y||timeout$;
$V \leftarrow \mathsf{Sign}(1/(J+s), R)$;
Calculate ZKPOK Φ' of (J, u, s, σ, x) such that:
  $y = g^x$
  $0 \leq J < n$
  $S = F_s(J)$
  $\text{VerifySig}(pk_{\mathcal{B}}, (u, s), \sigma) = true$
return (S, Φ', R, V, x, y)

We describe how a user calculates a coin in Algorithm 3.4.1. All the global parameters are the same as before. The unendorsed coin is (S, Φ', R, V, y) and the endorsement is x. The homomorphic one-way function $\phi : \mathbb{Z}_q \rightarrow G$ is defined as $\phi(x) = g^x$.

The function Sign ties $R = (pk_{\mathcal{M}}||contract||y||timeout)$ to the serial number of the coin. If we use the Dodis-Yampolskiy PRF, then $S = F_s(J) = g^{1/(1+s+J)}$. We can sign R using a discrete logarithm based signature scheme such as Schnorr [Sch91] or DSS [Kra99] with $1/(1+s+J)$ as the secret key and S as the verification key. Alternatively, we can use the even more efficient BLS [BLS01] signature: The ZKPOK for the Dodis-Yampolskiy PRF requires a bilinear map $e : \bar{G} \times \bar{G} \rightarrow G$ and publishing a proof $\pi = \bar{g}^{1/(1+s+J)}$. We can sign R using $1/(1+s+J)$ as the secret key and π as the verification key.

The OnLineSpend protocol works as follows: The user invokes SplitOnLineCoin to generate an endorsable coin (S, Φ', R, V, x, y) and gives (S, Φ', R, V, y) to the merchant. The merchant verifies the unendorsed coin and takes it to the bank to reserve the coin until *timeout*. The bank verifies that the unendorsed coin is valid. Then it checks if $S \in L \cup L'$, where L is the list of previously spent coins and L' is the list of temporarily locked serial numbers; if yes, this means the user is trying to double-spend and the bank informs the merchant not to accept the coin. If the unendorsed coin passes the test, the bank notifies the merchant and adds S to L'. If the merchant deposits the endorsed coin before *timeout* then the bank transfers S from L' to L. Otherwise, merchant returns to deposit the coin, the bank simply removes S from L'. It is the merchant's responsibility to make sure the fair exchange resolves before the *timeout* occurs and the user's responsibility not to create any unendorsed coins with the same serial number until after *timeout*.

In off-line endorsed e-cash, a malicious TTP can trick a user into double-spending by falsely claiming a fair exchange terminated unsuccessfully. As a consequence, when the user tries spending the wallet coin a second time, the bank learns the user's identity and may even trace all of the other coins the user spent. Even if the user later produces a certificate from the TTP stating that the first exchange was supposed to be aborted, the user's privacy is already compromised. In on-line e-cash, this is no longer an issue. The user's identity can never be revealed because there is no double-spending equation. If the bank gets an endorsed coin from a fair exchange that a TTP claimed was aborted, then the user can resolve the issue anonymously by publishing the signed abort certificate.

On-line endorsed e-cash has roughly the same communication cost as off-line endorsed e-cash. The biggest difference is that the bank must store serial numbers that *might* be used.

4 Endorsed E-Cash Protocols

Endorsed e-cash is better than standard e-cash because the lightweight structure of endorsements lends it to many nice protocols. In this section, we describe three such protocols: optimistic fair exchange of a single endorsed e-coin for digital goods and services, efficient fair exchange of multiple coins, and threshold secret sharing of endorsements.

4.1 Optimistic Fair Exchange of E-Cash for Digital Goods

We want to be able to exchange e-cash for digital content. We want the exchange to be fair: either the user gets the digital content and the merchant gets an e-coin or neither of them get anything. Asokan, Shoup and Waidner [ASW00] present an optimistic fair exchange protocol for exchanging digital signatures for digital goods with the help of a trusted third party (TTP). It is optimistic in the sense that the TTP gets involved only if either the user or the merchant violate the protocol.

Suppose user Alice has a signature and merchant Bob has the goods. The Asokan et al. protocol requires Alice to reduce the promise of her signature to the promise of a homomorphic pre-image. This is pre-

2007 IEEE Symposium on Security and Privacy(SP'07)
0-7695-2848-1/07 $20.00 © 2007 IEEE

cisely what endorsed e-cash does; to deposit an unendorsed coin $(\phi, y, coin')$, Bob must get $\phi^{-1}(y)$ from Alice. Asokan et al. propose a way to reduce e-cash, however, their method creates *linkable* coins while SplitCoin (see Section 3.2) generates *independent* coins.

Security: The Asokan et al. optimistic fair exchange protocol requires a tagged, CCA2 secure, verifiable encryption scheme for encrypting the pre-image of ϕ. In Section 3.2, we use $\phi(a,b,c) = h_1^a h_2^b g^c$; thus, we should use the Camenisch and Shoup [CS03] verfiable encryption scheme. Its security is based on the Paillier Assumption, and the length of the proof is optimal. We assume that ϕ is a one-way function, so we require the discrete logarithm assumption.

As long as the TTP is honest, the exchange will be fair. A dishonest TTP can cheat either party. Worse, a malicious TTP can trick a user into double-spending by falsely claiming that the exchange aborted. When the user retries spending the wallet coin, the bank learns the user's identity and may even trace the user's other coins. Therefore, we require the TTP to give the user and merchant signed "abort" certificates. A malicious TTP can still compromise the user's privacy, but at least the certificate lets the user prove his innocense and implicate the TTP.

4.2 Paying Multiple Coins

Suppose a merchant is selling a car for 19,995 e-coins (an e-coin can be worth a dollar, or some other amount if the system supports different denominations). If a user wants to do a fair exchange, she must verifiably encrypt 19,995 endorsements. Creating and verifying the ciphertexts is computationally expensive. Worse, if the trusted third party becomes involved, it must store *all* of the verifiable encryptions and their tags.

We can significantly reduce the cost of the fair exchange. Examine the unendorsed coin (S', T', Φ', R, y) from Section 3.2. The value $y = \phi(x_1, x_2, x_3)$, where (x_1, x_2, x_3) is the endorsement. A fair exchange of one coin for the car trades the opening of y for the opening of some value K. A fair exchange of n unendorsed coins trades the opening of $(y^{(0)}, \ldots, y^{(n-1)})$ for the opening of K. Because ϕ is really a Pedersen commitment, we can use a Pedersen VSS [Ped92] style algorithm to reduce opening all the $y^{(i)}$ to just opening $y^{(0)}$.

Setup: We will use the same public parameters as the endorsed e-cash system in Section 3.2. For notational convenience, we will use (g_1, g_2, g_3) instead of (h_1, h_2, g) (recall that these are three generators of G whose discrete logarithm representation relative to each other is unknown; we assume the discrete logarithm problem is hard in G). Therefore, $\phi(a,b,c) = g_1^a g_2^b g_3^c$.

User Promise: The user makes n new endorsable coins $(S'^{(i)}, T'^{(i)}, \Phi'^{(i)}, R^{(i)}, (x_1^{(i)}, x_2^{(i)}, x_3^{(i)}), y^{(i)})$, for $i \in [0, n-1]$. The user calculates three polynomials f_1, f_2, f_3 of degree $n-1$, such that $\forall i \in [0, n-1], \forall j \in \{1,2,3\} : f_j(i) = x_j^{(i)}$ (this is a simple interpolation). Let set $A = \{0, \ldots, n-1\}$. The user calculates $n-1$ new points $p_j^{(i)}$ on f_1, f_2 and f_3, as follows:

$$\forall i \in [n, 2n-2], \forall j \in \{1,2,3\} :$$
$$p_j^{(i)} = f_j(i) = \sum_{a \in A} f_j(a) \prod_{\substack{b \in A \\ b \neq a}} \frac{i-a}{b-a}$$

The user gives the merchant the n unendorsed coins and $\{p_j^{(i)} : i \in [n, 2n-2], j \in \{1,2,3\}\}$.

Merchant Verifies: The merchant gets n unendorsed coins $(S'^{(i)}, T'^{(i)}, \Phi'^{(i)}, R^{(i)}, y^{(i)})$, for $i \in [0, n-1]$, and uses the $\Phi'^{(i)}$ to verify their validity. Then the merchant checks that the openings of the $y^{(i)}$ are on the same polynomials as the $p_j^{(i)}$. He does not need to know the openings for this! Let set $B = \{n, \ldots, 2n-2\}$. The merchant accepts only if:

$$\forall i \in [1, n-1] :$$
$$y^{(0)} = (y^{(i)})^{\prod_{b \in B} \frac{i}{i-b}} \prod_{j=1}^{3} (g_j)^{\sum_{a \in B} \frac{a}{a-i} p_j^a \prod_{\substack{b \in B \\ b \neq a}} \frac{a}{a-b}}$$

Fair Exchange: The merchant and the user conduct an optimistic fair exchange of the opening of $y^{(0)}$ for the opening of K. The merchant learns $\phi^{-1}(y^{(0)}) = (x_1^{(0)}, x_2^{(0)}, x_3^{(0)})$. If the exchange fails, the user must throw out the unendorsed coins.

Reconstruct: The merchant uses $(x_1^{(0)}, x_2^{(0)}, x_3^{(0)})$ and $\{p_j^{(i)} : i \in [n, 2n-2], j \in \{1,2,3\}\}$ to learn the openings of $y^{(1)}, \ldots, y^{(n-1)}$. He sets $C = \{0, n, \ldots, 2n-2\}$, $p_j^{(0)} = x_j^{(0)}$, and calculates:

$$\forall i \in [1, n-1], \forall j \in \{1,2,3\} :$$
$$x_j^{(i)} = f_j(i) = \sum_{a \in C} f_j(a) \prod_{\substack{b \in C \\ b \neq a}} \frac{x-a}{b-a}$$

Theorem 4.1 (Multi-coin fair exchange is secure)**.** *Suppose a group of malicious merchants asks a group of honest users to engage in an arbitrary number of fair exchange protocols. The users have access to an unlimmitted number of withdrawals from the bank. The merchants can terminate the fair exchanges at any point. If the users endorse N e-coins and the merchants withdraw M e-coins from the bank, then the merchants can deposit at most $N + M$ e-coins.*

Before proving Theorem 4.1, we first prove in Lemma 4.2 that if a merchant endorses an e-coin during

2007 IEEE Symposium on Security and Privacy(SP'07)
0-7695-2848-1/07 $20.00 © 2007 IEEE

a single run of a *failed* multi-coin exchange, then he can calculate discrete logarithms. Then we use Lemma 4.2 to show that if the merchants manage to deposit more coins than the users intended to give him (and that they withdrew from the bank), then the merchants violate either the security of the endorsed e-cash scheme or the discerete logarithm assumption.

Lemma 4.2. *Let $(x_1^{(0)}, x_2^{(0)}, x_3^{(0)}), \ldots, (x_1^{(n-1)}, x_2^{(n-1)}, x_3^{(n-1)})$ be numbers in $\mathbb{Z}_p$ selected at random, and let g_1, g_2, g_3 be generators of a group G. We define the function $\phi(a,b,c) = g_1^a g_2^b g_3^c$ and calculate $y^{(0)}, \ldots, y^{(n-1)}$, such that $y_i = \phi(x_1^{(i)}, x_2^{(i)}, x_3^{(i)})$. In addition, the $x_j^{(i)}$ define three polynomials f_1, f_2, f_3 such that $f_j(i) = x_j^{(i)}$ for $0 \leq i \leq n-1$. We calculate $n-1$ points on each of these three polynomials: $\{p_j^{(i)} = f_j(i) : i \in [n, 2n-2], j \in \{1,2,3\}\}$. Suppose there exists an adversary that on input G, g_1, g_2, g_3, $(y^{(0)}, \ldots, y^{(n-1)})$, and $\{p_j^{(i)} = f_j(i) : i \in [n, 2n-2], j \in \{1,2,3\}\}$ outputs (a,b,c) such that $y^{(i)} = \phi(a,b,c)$, for some $i \in [0, n-1]$. Then we can use this adversary to calculate discrete logarithms in G.*

Proof. We construct a reduction that uses the adversary from Lemma 4.2 as a black-box to calculate discrete logarithms. The reduction gets y as input. Suppose (x_1, x_2, x_3) is the opening of y; the reduction does not know these values, but it constructs three polynomials f_1, f_2, f_3 so that $f_j(0) = x_j$. First the reduction randomly chooses $3(n-1)$ numbers in $\mathbb{Z}_p$: $\{p_j^{(i)} : i \in [n, 2n-2], j \in \{1,2,3\}\}$; these will be random points that, along with the (unknown) opening of y, define the polynomials f_1, f_2, f_3. Then the reduction calculates $y^{(1)}, \ldots, y^{(n-1)}$. Let $S = [n, 2n-1]$, then:

$$\forall i \in [1, n-1]:$$

$$y^{(i)} = y^{\prod_{b \in S} \frac{i}{b}} \prod_{j=1}^{3} (g_j)^{\sum_{a \in S} p_j^a \prod_{\substack{b \in S \cup \{0\} \\ b \neq a}} \frac{i-a}{b-a}}$$

The reduction passes $(y, y^{(1)}, \ldots, y^{(n-1)})$ and $\{p_j^{(i)} : i \in [n, 2n-2], j \in \{1,2,3\}\}$ to the black-box. The black-box responds with an opening to one of the $y^{(i)}$. From this the reduction can interpolate the polynomials and open y. □

Proof of Theorem 4.1. We now show that no merchant can take advantage of the multi-coin fair exchange protocol to deposit more coins than the honest users intended to give him. Suppose a group of dishonest merchants, after withdrawing M e-coins and running a number of multi-coin fair exchanges in which only N coins should be endorsed, manages to deposit more than $M+N$ coins. Then we can construct a reduction that uses the merchants as a black-box to either break the balance or anonymity properties of the endorsed e-cash scheme, or to calculate discrete logarithms.

The reduction gets y as input and needs to output (x_1, x_2, x_3) such that $y = h_1^{x_1} h_2^{x_2} g^{x_3}$. The reduction sets up an endorsed e-cash system, using (h_1, h_2, g) as the public parameters. It also uses $\mathcal{S}_{ZK}$, the simulator for the zero-knowledge system Φ' to create $(params_{ZK}, auxsim_{ZK})$ and $\mathcal{S}_{CHL}$, the simulator for the CHL e-cash system to create $(params_{CHL}, auxsim_{CHL})$.

The reduction runs multi-coin fair exchanges with the merchants. In one of those exchanges (the reduction chooses which one at random), the reduction inserts y into an unendorsed coin. Suppose a merchant wants n coins. Then the reduction prepares the input to the merchant as follows: It asks $\mathcal{S}_{CHL}$ to create n e-coins $(S^{(i)}, T^{(i)}, \Phi^{(i)}, R^{(i)})$ (the reduction runs Withdraw and Spend the appropriate amount of times). Then the reduction uses y to create an unendorsed coin. It randomly chooses r_1 and r_2 and calculates $S' = Sg^{r_1}$ and $T' = Tg^{r_2}$ (we need to blind S and T; we don't know any valid openings of y, but for any r_1 and r_2 we choose, there exists *some* r_3 such that $\phi(r_1, r_2, r_3) = y$). Then it uses $\mathcal{S}_{ZK}$ to generate a fake proof Φ' such that an honest merchant would accept the unendorsed coin $coin'^{(0)} = (S'^{(0)}, T'^{(0)}, \Phi'^{(0)}, R'^{(0)}, y)$. Next the reduction chooses the random points on three polynomials: $\{p_j^{(i)} : i \in [n, 2n-2], j \in \{1,2,3\}\}$. Finally, the reduction chooses the appropriate $y^{(1)}, \ldots, y^{(n-1)}$ (using the same method as in the proof of Lemma 4.2) and uses $\mathcal{S}_{CHL}$ and $\mathcal{S}_{ZK}$ to create the unendorsed coins $coin'^{(1)}, \ldots, coin'^{(n-1)}$. The reduction gives $coin'^{(0)}, \ldots, coin'^{(n-1)}$ and $\{p_j^{(i)} : i \in [n, 2n-2], j \in \{1,2,3\}\}$ to the merchant.

Eventually, the merchants output a list of more than $M+N$ coins for deposit. At least one of these coins must be fake. If it is an entirely new coin then the merchant violated the balance property of the endorsed e-cash scheme. The only other possibility is that the coin was from a terminated multi-coin fair exchange. With non-negligible probability, the reduction would have inserted y into that fair exchange. In this case, by Lemma 4.2, the merchants violated the discrete logarithm assumption. If the merchants *fail* to output more than $M+N$ coins, then the merchants violated anonymity because they distinguished the simulator from real users (this can be shown with a straightforward reduction). □

4.3 Threshold Endorsed E-cash

Sometimes, such as in our onion routing example, we want to require the merchant to acquire several en-

2007 IEEE Symposium on Security and Privacy(SP'07)
0-7695-2848-1/07 $20.00 © 2007 **IEEE**

dorsements before reconstructing an e-coin. In this section, we construct a threshold endorsed e-cash system where the merchant needs to get m out of n possible endorsements.

An unendorsed coin consists of (S', T', Φ', R, y), where $y = \mathsf{Ped}(x_1, x_2; x_3)$. We can use Pedersen Verifiable Secret Sharing [Ped92] to create shares of the endorsement. For notational convenience, we use (g_1, g_2, g_3) instead of original parameters (h_1, h_2, g) in Section 3.2.

To share (x_1, x_2, x_3), the user generates three random polynomials f_1, f_2, f_3 of degree $m-1$ such that $f_j(0) = x_j$. The user stores a secret vector of n points on the polynomial; these points are the endorsements. The user gives the merchant commitments to the coefficients that define the polynomials. Once the merchant learns m points on the polynomials, he can recover (x_1, x_2, x_3) and endorse the coin. Algorithm 4.1 describes how the user creates a threshold endorsable coin.

Algorithm 4.1: SplitCoinMN

Input: $pk_{\mathcal{M}} \in \{0,1\}^*$ merchant's public key, $contract \in \{0,1\}^*$
User Data: u private key, g^u public key, (s, t, σ, J) a wallet coin
$(S', T', \Phi', R, (x_1, x_2, x_3), y) \leftarrow \mathsf{SplitCoin}(pk_{\mathcal{M}}, contract)$;
$a_{j,0} \leftarrow x_j, \forall j \in \{1,2,3\}$;
$a_{j,k} \leftarrow \mathbb{Z}_p, \forall j \in \{1,2,3\}, \forall k \in [1, m-1]$;
$Z \leftarrow \{Z_k = \prod_{j=1}^{3} g_j^{a_{j,k}} : k \in [0, m-1]\}$;
$X \leftarrow \{X_j^{(i)} = \sum_{k=0}^{m-1} a_{j,k} i^k : j \in \{1,2,3\}, i \in [0,n]\}$;
return (S', T', Φ', R, X, Z)

In $\mathsf{MNSpend}$, the user gives the merchant the threshold unendorsed coin (S', T', Φ', R, Z) and stores the endorsement X. The merchant needs to verify the unendorsed coin: he uses uses Z, a commitment to the polynomials' coefficients, to calculate Y, a commitment to points on the polynomials: $Y = \{Y^{(i)} = \prod_{k=0}^{m-1} (Z_k)^{i^k} : i \in [0,n]\}$. The merchant sets $y = Y^{(0)}$ and verifies Φ' in the usual way.

Now the merchant needs to get m endorsements. The user has n endorsements: $\{(X_1^{(i)}, X_2^{(i)}, X_3^{(i)}) : i \in [1,n]\}$. They can use the homomorphic one-way function $\phi(a,b,c) = g_1^a g_2^b g_3^c$ to do an optimistic fair exchange because $\phi(X_1^{(i)}, X_2^{(i)}, X_3^{(i)}) = Y^{(i)}$ (remember, Φ' proves the $Y^{(i)}$ are correct). In $\mathsf{MNReconstruct}$, the merchant uses the m points to interpolate the polynomials and learn (x_1, x_2, x_3).

Security: this is a straightforward application of Pedersen VSS. The user creates n verifiable shares of the secret (x_1, x_2, x_3) and gives the merchant the standard verification vector. Each endorsement is a share of the secret.

5 Conclusion

We have shown how to perform truly fair exchange of off-line and on-line e-cash for digital goods and services. We provide a new protocol for efficiently exchanging multiple e-coins simultaneously; this protocol can be applied to fair exchange of any secret that lends itself to Pedersen commitments. Our threshold endorsed e-cash allows exchanging a single e-coin for multiple goods and services. By reducing the exchange of e-cash to the exchange of lightweight endorsements, we make it possible to apply many (efficient) standard cryptographic techniques to e-commerce.

Acknowledgements

Part of Jan Camenisch's work reported in this paper is supported by the European Commission through the IST Programme under Contracts IST-2002-507932 ECRYPT and IST-2002-507591 PRIME. The PRIME projects receives research funding from the European Communitty's Sixth Framework Programme and the Swiss Federal Office for Education and Science. Anna Lysyanskaya is supported by NSF Grant CNS-0627553. Mira Meyerovich is supported by a U.S. Department of Homeland Security Fellowship and NSF Grant CNS-0627553. All opinions expressed in this paper are the authors' and do not necessarily reflect the policies and views of EC, DHS, and NSF.

References

[ADS03] Alessandro Acquisti, Roger Dingledine, and Paul Syverson. On the economics of anonymity. *Financial Cryptography*, pages 84–102, 2003.

[ASW97] N. Asokan, Matthias Schunter, and Michael Waidner. Optimistic protocols for fair exchange. In *Proc. 4th ACM Conference on Computer and Communications Security*, pages 6–17. ACM press, 1997.

[ASW00] N. Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in Communications*, 18(4):591–610, April 2000.

[BB04] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity based encryption without random oracles. In *Advances in Cryptology — EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 223–238. Springer-Verlag, 2004.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *Proceedings of ASIACRYPT2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer-Verlag, 2001.

[BP97] Niko Barić and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In Walter Fumy, editor, *Advances in Cryptology — EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 480–494. Springer Verlag, 1997.

[BP02] Mihir Bellare and Adriana Palacio. Gq and schnorr identification schemes: Proofs of security against impersonation under active and concurrent attacks. In Moti Yung, editor, *Advances in Cryptology — CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 162–177. Springer Verlag, 2002.

[Bra93a] Stefan Brands. An efficient off-line electronic cash system based on the representation problem. Technical Report CS-R9323, CWI, April 1993.

[Bra93b] Stefan Brands. Untraceable off-line cash in wallets with observers. manuscript, CWI, 1993.

[Bra93c] Stefan Brands. Untraceable off-line cash in wallets with observers. In Douglas R. Stinson, editor, *Advances in Cryptology — CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 302–318, 1993.

[CFN90] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In Shafi Goldwasser, editor, *Advances in Cryptology — CRYPTO '88*, volume 403 of *Lecture Notes in Computer Science*, pages 319–327. Springer Verlag, 1990.

[Cha81] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, February 1981.

[Cha83] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology — Proceedings of CRYPTO '82*, pages 199–203. Plenum Press, 1983.

[Cha84] David Chaum. Blind signature systems. In David Chaum, editor, *Advances in Cryptology — CRYPTO '83*, page 153. Plenum Press, 1984.

[CHK+06] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysysanskaya, and Mira Meyerovich. How to win the clone wars: Efficient periodic n-times anonymous authentication. In *ACM CCS*, pages 201–210, 2006.

[CHL05] Jan Camenisch, Susan Hohenberger, and Anna Lysysanskaya. Compact e-cash. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT*. lncs, 2005.

[CHL06] Jan Camenisch, Susan Hohenberger, and Anna Lysysanskaya. Balancing accountability and privacy using e-cash. In *Security and Cryptography for Networks – SCN*. lncs, 2006.

[CL02] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In Giuseppe Persiano, editor, *Security in communication networks*, volume 2576 of *Lecture Notes in Computer Science*, pages 268–289. Springer Verlag, 2002.

[CL04] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Advances in Cryptology — CRYPTO 2004*, Lecture Notes in Computer Science. Springer Verlag, 2004.

[CL05] Jan Camenisch and Anna Lysyanskaya. A formal treatment of onion routing. In Victor Shoup, editor, *Advances in Cryptology — CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 169–187. Springer-Verlag, August 2005.

[Coh03] Bram Cohen. Incentives build robustness in bittorrent. In *Procceedings of the 1st Workshop on Economics of Peer-to-Peer Systems, Berkeley, CA*, 2003.

[CP93] David Chaum and Torben Pryds Pedersen. Transferred cash grows in size. In Rainer A.

2007 IEEE Symposium on Security and Privacy(SP'07)
0-7695-2848-1/07 $20.00 © 2007 IEEE

Rueppel, editor, *Advances in Cryptology — EUROCRYPT '92*, volume 658 of *Lecture Notes in Computer Science*, pages 390–407. Springer-Verlag, 1993.

[CPS94] Jan L. Camenisch, Jean-Marc Piveteau, and Markus A. Stadler. Blind signatures based on the discrete logaritm problem. In Alfredo De Santis, editor, *Advances in Cryptology — EUROCRYPT '94*, volume 950 of *Lecture Notes in Computer Science*, pages 428–432. Springer Verlag Berlin, 1994.

[CS03] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In Dan Boneh, editor, *Advances in Cryptology — CRYPTO 2003*, Lecture Notes in Computer Science. Springer Verlag, 2003. To appear.

[CTS95] B. Cox, J. D. Tygar, and M. Sirbu. Netbill security and transaction protocol. In *Proceedings of the First Usenix Workshop on Electronic Commerce*, pages 77–88, 1995.

[DDM03] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 2003.

[DY05] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In *Proceedings of the Workshop on Theory and Practice in Public Key Cryptography*, volume 3386 of *Lecture Notes in Computer Science*, pages 416–431, 2005.

[FTY96] Yair Frankel, Yiannis Tsiounis, and Moti Yung. "Indirect discourse proofs:" Achieving efficient fair off-line e-cash. In Kwangjo Kim and Tsutomu Matsumoto, editors, *Advances in Cryptology — ASIACRYPT '96*, volume 1163 of *Lecture Notes in Computer Science*, pages 286–300. Springer Verlag, 1996.

[FY92] Matthew Franklin and Moti Yung. Towards provably secure efficient electronic cash. Technical Report TR CUSC-018-92, Columbia University, Dept. of Computer Science, April 1992. Also in: Proceedings of ICALP 93, Lund, Sweden, July 1993, volume 700 of LNCS, Springer Verlag.

[Jak95] Markus Jakobsson. Ripping coins for a fair exchange. In *Advances in Cryptology — EUROCRYPT '95*, volume 921, pages 220–230. Springer Verlag, 1995.

[Kra99] Hugo Krawczyk. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology — CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer Verlag, 1999.

[McC01] Jim McCoy. Mojo nation responds, 2001.

[Mic97] Silvio Micali. Certified e-mail with invisible post offices. Presentation at the 1997 RSA Security Conference, 1997.

[MSK02] Shigeo Mitsunari, Ryuichi Sakai, and Masao Kasahara. A new traitor tracing. In *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, volume E85-A, No. 2, pages 481–484, 2002.

[NSN05] Lan Nguyen and Rei Safavi-Naini. Dynamic k-times anonymous authentication. In *Applied Cryptography and Network Security*, volume 3531 of *Lecture Notes in Computer Science*, pages 318–333, New York, 2005.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite residuosity classes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–239. Springer Verlag, 1999.

[Ped92] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer Verlag, 1992.

[RWW05] Michael Reiter, XiaoFeng Want, and Matthew Wright. Building reliable mix networks with fair exchange. In *Applied Cryptography and Network Security: Third International Conference*, pages 378–392. Lecture Notes in Computer Science, June 2005.

[Sch91] Claus P. Schnorr. Efficient signature generation for smart cards. *Journal of Cryptology*, 4(3):239–252, 1991.

[SPC95] Markus Stadler, Jean-Marc Piveteau, and Jan Camenisch. Fair blind signatures. In Louis C. Guillou and Jean-Jacques Quisquater, editors, *Advances in Cryptology — EUROCRYPT '95*, volume 921 of *Lecture Notes in Computer Science*, pages 209–219. Springer Verlag, 1995.

[TS04] Isamu Teranishi and Kazue Sako. k-times anonymous authentication with a constant proving cost. In *Public Key Cryptography – PKC 2006*, volume 3958 of *Lecture Notes in Computer Science*, pages 525–542, New York, NY, 2004.

[Tsi97] Yiannis S. Tsiounis. *Efficient Electonic Cash: New Notions and Techniques*. PhD thesis, Northeastern University, Boston, Massachusetts, 1997.

[Wei05] Victor K. Wei. More compact e-cash with efficient coin tracing. Available at: `http://eprint.iacr.org/2005/411`, 2005.

2007 IEEE Symposium on Security and Privacy(SP'07)
0-7695-2848-1/07 $20.00 © 2007 IEEE