Combating Double-Spending Using Cooperative P2P Systems

Ivan Osipkov Eugene Y. Vasserman Nicholas Hopper
Yongdae Kim
Computer Science & Engineering, University of Minnesota, Minneapolis, MN 55455
{osipkov,eyv,hopper,kyd}@cs.umn.edu

Abstract

An electronic cash system allows users to withdraw coins, represented as bit strings, from a bank or broker, and spend those coins anonymously at participating merchants, so that the broker cannot link spent coins to the user who withdraws them. A variety of schemes with various security properties have been proposed for this purpose, but because strings of bits are inherently copyable, they must all deal with the problem of double-spending. In this paper, we present an electronic cash scheme that introduces a new peer-to-peer system architecture to prevent double-spending without requiring an on-line trusted party or tamper-resistant software or hardware. The scheme is easy to implement, computationally efficient, and provably secure. To demonstrate this, we report on a proof-of-concept implementation for Internet vendors along with a detailed complexity analysis and selected security proofs.

1. Introduction

The physical world has untraceable, transferable, double-spending-protected cash; the current Internet does not. There are systems that allow on-line payments using credit card transactions or bank accounts, but these 1) are not cost-effective for small transactions as the fees paid by merchants for credit card transactions are typically higher than a dollar, 2) do not provide anonymity, allowing credit card companies to track spending and giving merchants access to sensitive credit card and/or bank information, and 3) do not offer security.[1] The credit card business model evolved before the advent of ubiquitous networked communication and cheap computing, and has retained the old pricing structure, vastly over-estimating communication and processing costs. While Rivest defines micro-payments as payments of less than $10 [36], the market has shown that merchants are willing to accept credit card transactions as low as two dollars, and in case of large-volume transactions, such as Apple's iTunes music store, as low as one dollar. This paper deals with what we call "mini-payments" for values similar to typical physical coins, occupying roughly the gap between fractional cent payments that incur more cost than their value and payments which can be handled profitably through the credit card infrastructure.

The use of untraceable and anonymous mini-payments could enable a number of interesting and new on-line applications, and could be applied to alter existing business models. Advertising-supported web sites could remove ads entirely and charge a penny or so for access; long-term site subscriptions could be replaced with short-term; donation-dependant sites could be advertisement-free, relying on numerous small donations; software "bundles" could be "unbundled". As the psychological barrier when donating smaller amounts is lower, the potential for donation-generated income from mini-payments is likely higher than that from large donations from a smaller number of users, benefiting vendors providing free software [36] who would welcome donations of any size. Price discrimination is also not an issue in case of donations. Such business models only work if mini-payment processing transactions are almost free and the mini-payment themselves are easy and intuitive for customers to use. The "killer app" for mini-payments may be unclear, but if the "long tail" argument holds for digital goods, surely it holds for digital cash!

There are many potential benefits to such a system, but several problems remain to be solved. One major attack on electronic currency is double-spending, where a user may spend an electronic coin more than once. Unless the merchant accepting the coin verifies each coin immediately, double-spending poses a significant threat. Individual coins may be worth little, but the danger of large groups doing concurrent double-spending using the same coin is non-trivial. Many e-cash schemes have been suggested in the past, but all of them either require the presence of an on-line third party, tamper-proof hardware or client accounts at the bank. Tamper-proof hardware creates a significant hurdle for proliferation of such a scheme, since most current machines have none. Requiring an on-line third party creates a single point of failure, and creates

[1] Consider the many public cases where stored credit card information has been compromised [20]. Internet transactions also present more risk and consequently the fees are higher. Also, outside the US, customers are liable for fraud committed with their credit cards. The success of *Ukash* [1] in EU countries demonstrates that many users are willing to go through extra hurdles to avoid credit card use on the Internet.

administrative and equipment expenses (especially during peak hours). Moreover, it is not always clear which entity should be endowed with such a role.

Foregoing on-line detection, however, introduces delay in double-spending detection (until the coins are deposited) and therefore requires clients to leave security deposits or credit cards at the bank. Leaving credit card information is a deterrent to proliferation, which grows stronger every day due to constantly publicized attacks on private information and compromise of home computers. E-cash without on-line double spending detection exacerbates these problems: even if the credit card information is secure against attackers, the security of the coins themselves can create significant problems. This is because if the coin itself is stolen by an attacker, it can be used freely to double-spend; in the end, the victim will have to cover the damage.

If we demand double-spending to be non-prosecutable, a natural requirement is to make e-cash completely anonymous and untraceable: this would shield clients against liability for fraud committed with the coins and also allow easy transfer of coins to others. *However, in this case, real-time double-spending detection becomes a critical requirement.*[2]

Overview of the paper. The primary contribution of this paper is development of a lightweight, provably secure distributed anonymous e-cash protocol that does not require a trusted on-line third party, tamper-proof hardware or security deposits, and provides real-time double-spending detection. This protocol is presented in Sections 4 and 5. We demonstrate the efficacy of this protocol in several ways: 1) derivation of security requirements in Section 3 and security proof in Section 6, 2) Analysis of the computational and communication complexity in Section 7, 3) a prototype implementation and experimental results in Section 7.

Our system is a "bearer" system, where the client holds a bit-string representing the coin. The coin is not bound to anyone except the broker who exchanges it for real-world cash. Due to our real-time double-spending detection scheme, we do not require tamper-proof hardware. This system design is a three-party model, with the broker as a dedicated (but not necessarily on-line) server, the merchant as a drop-in module for an existing web server, and the client as a browser plug-in. The client purchases coins from the broker using a dedicated web interface and the browser plug-in stores the coins in a file, where each coin is assigned non-malleably to a *witness*(es) selected randomly from all merchants participating in the mini-payment network. A web server taking mini-payments signals the service availability to the client, who then displays the payment user interface (the mode of display depends whether the payment is optional or required to view content). To submit a mini-payment, the client contacts the merchant and transmits a coin. The merchant, in turn, submits the coin for signature by the coin's designated witness(es). If the witness(es) have seen the coin before, they can prove this to the merchant by extracting some secret information from two instances of the coin and the merchant would then reject the payment, thus providing immediate double-spending detection. If not, the witness(es) sign the coin and return it to the merchant, who then accepts the payment. Signed coins can be cashed at the broker at any time. Note that the coin contains a secret value that is not revealed to protect the coin from third-party theft. Instead, an efficient non-interactive proof of knowledge of that value is provided.

[2]Incidentally, absence of real-time double-spending detection can also create room for attacks using stolen credit cards: an attacker can buy a few coins using the stolen credit card (to stay under the radar) and then freely double-spend these coins; the credit card companies (or card owner) will have to cover the losses.

2. Related Work

E-cash should not be confused with *micro-payments*, which deal with payments as low as a fraction of a cent [25, 38, 33, 32, 31, 24, 23, 18, 22, 21, 4] and require optimization for performance. Thus, the more promising schemes use a probabilistic approach [25, 23] when deciding whether to charge a client: the resulting inaccuracy, though, may not be forgivable in case of larger and less frequent payments. E-cash is also different from electronic cheques, which work just like like normal cheques – they are not anonymous and require overhead similar to credit card processing.

The idea of untraceable electronic cash was first introduced by Chaum [11], who used blind signatures to ensure that the e-cash cannot be traced back to the client. This initial proposal required an on-line broker to clear coins before merchants would provide their services, to protect against double-spending. The first off-line untraceable electronic cash was proposed by Chaum *et. al.* [12]; in this scheme, each coin contains a hidden reference to the coin owner: if the coin is spent once it is untraceable, while spending a coin twice allows the broker to extract the identity hidden inside the coin. The scheme in [12] requires clients to set up accounts at the broker and leave a security deposit or credit card. The scheme also uses an inefficient cut-and-choose technique to verify correctness of the blind information. The first efficient untraceable off-line e-cash scheme was suggested by Brands [7], and further improved by Chan *et. al.* [9, 10]. Brands' scheme also incorporated the idea of "wallets with observers" [13], in which a tamper-proof device used by the client offered a first-line real-time defense against double-spending. Several properties have been explored in successive works, including "divisibility" of coins [28, 27, 30, 15], compactness [8], tracing of coins spent illegitimately [17, 39], and coin transferability [14].

E-Cash can be used not only in traditional customer-merchant systems but also in P2P systems. *PPay* [40] uses e-cash as a payment system for P2P systems, leveraging the fact that peers are clients and merchants at the same

time: thus, clients can pay with the (transferable) coins that they obtain from selling their own goods, minimizing the number of interactions with the bank/broker. The *WhoPay* scheme [37] extends the idea of *PPay* and ensures that coins are anonymous as well as distributing broker load to the peers themselves. In addition, the paper suggests a mechanism for *real-time double-spending detection* by which the P2P system is used as a distributed database for spent coins and queried using a DHT routing layer such as Chord [34]. Hoepman [19] discusses the same idea in more depth and evaluates different scenarios for the location of stored, spent coins. However, neither approach can provide hard guarantees against double-spending, especially when some fraction of P2P nodes are compromised: the distributed database cannot be fully trusted unless secure routing and honesty of peers are guaranteed and can only support probabilistic guarantees. A similar approach, which we call the *witness approach*, was successfully applied to fairness enforcement in P2P file-archiving systems [29]. The witness approach provides probabilistic guarantees as well but also ensures security against targeted attacks since witnesses change dynamically and at random times. In this paper, we adapt the witness approach [29] to ensure real-time double-spending prevention: however, Section 4 outlines several *non-trivial changes applied to ensure that the system provides hard, rather than probabilistic, guarantees.*

3. Basic Requirements and Observations

A generic e-cash system consists of 1) the *bank* that clears credit/debit card or bank payments but may not know anything about e-cash; 2) *brokers* which interact with the bank and are involved in the printing and redemption of coins; 3) *on-line merchants* who accept e-cash coins as payment for services and cash them using the broker; and 4) *clients* who obtain e-cash from the broker (or through other means) and then use it to buy services from merchants. A mini-payment scheme suitable for widespread adoption should satisfy the following requirements:

Decentralized environment. As in *off-line e-cash systems* [12], there should be no centralized on-line trusted party required to participate in transactions. In particular, it should be possible to spend coins and prevent double-spending even if the broker and bank are off-line.

No Tamper-proof devices. As tamper-proof devices in general hamper proliferation of e-cash schemes, the system should not require them.

Untraceability. The bare system design, unlike related work, should allow for full untraceability of purchased coins. In particular, double-spending should not leak any information about the coin owner.

Client Security Deposits. As the bare system should provide untraceability, there is no need for client security deposits at the bank and/or broker. Thus, unlike other off-line schemes, the client should not bear any responsibility with respect to purchased coins (except insofar as they have value to the client). In particular, a client may choose to buy coins using an on-line gift card without revealing identity.

Generic Security. The system should be secure against coin forgery/re-use/linkability and other generic e-cash attacks as discussed in Section 6. These security notions are generic to e-cash as defined by Chaum [12, 35].

Usability and Extendibility. The system should allow for *incorporation* of escrow mechanisms that allow tracing the coin owner. The system should be flexible enough to accommodate known off-line double-spending detection mechanisms.

If a client is untraceable, then there is a danger of double-spending. Thus *these requirements dictate that the system should also provide real-time double-spending detection*, or in other words *double-spending prevention*.

Since we will be leveraging the distributed nature of the merchant network, the following basic observations are in order. *First*, merchants are long-term members of the network, are legitimate, and can therefore set up accounts with the broker, leaving security deposits if necessary. *Second*, merchants are on-line most of the time and are generally well-maintained. The implication is that even if the merchant network is attacked, it will go back on-line within a few days. These assumptions are safe because on-line merchants can only make money if they remain on-line, and thus it is in their best interest to do so. *Third*, the merchants themselves can form a network to combat double-spending. We use these observations to construct the desired e-cash system, starting first with the high-level description below.

4. High-Level Description

Figure 1 provides an overview of the functionalities involved in the proposed e-cash system. To use the proposed system, merchants need to set up an account with the broker $\mathcal{B}$, by supplying their certified public key, credentials and bank accounts where e-cash should be deposited. Moreover, each merchant $\mathcal{M}$ leaves a security deposit in the form of cash or a credit card. This registration allows $\mathcal{M}$ to redeem coins obtained from clients, and it allows $\mathcal{B}$ to charge $\mathcal{M}$ for its misbehavior.

To obtain e-cash, clients have to contact $\mathcal{B}$ and buy coins using a credit card or bank account. To make the purchased coins untraceable, we employ cryptographic techniques to blind the *private* information of the coin before it is signed by $\mathcal{B}$ while the *public* information (such as expiration dates) stays unblinded. As a result, $\mathcal{B}$ will have no information about the coin itself, except the public information.

When a client goes to a merchant $\mathcal{M}$ and presents a coin, $\mathcal{M}$ needs to determine in real-time if the coin is being double-spent before providing the service: otherwise, either 1) $\mathcal{M}$ may erroneously provide service in exchange

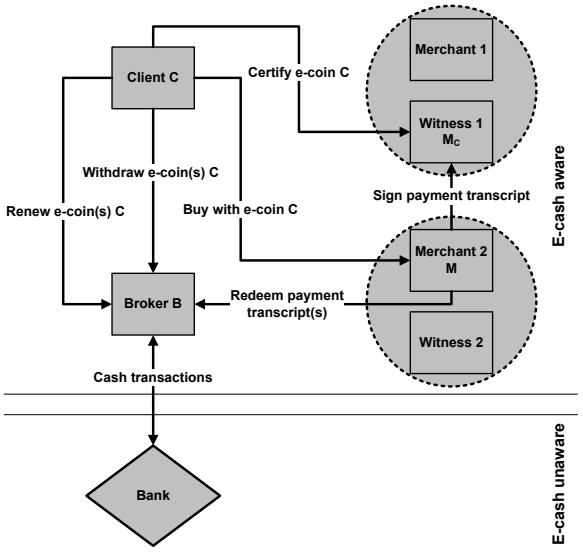

Figure 1. High-level view of the proposed E-cash system.

for an invalid coin (or uncashable during a later deposit), and with no entity that covers losses from fraud, $\mathcal{M}$ will not be paid by $\mathcal{B}$, or 2) $\mathcal{M}$ will have to delay service delivery, degrading the quality (or speed) of service. We achieve this as follows: during its creation, each coin is assigned in a random fashion to one of the merchants, which will serve as a *witness* for the validity of that coin. *Thus, each merchant can perform some witness service: witnesses for coins are chosen from the merchants*. Say, coin C is assigned to merchant $\mathcal{M}_C$ who serves as a witness for the coin: whenever another merchant $\mathcal{M}$ obtains the coin C from the client, $\mathcal{M}$ has to contact $\mathcal{M}_C$ and obtain a signature on the payment transcript, which testifies that the coin has not been used before. Without a signature from $\mathcal{M}_C$, $\mathcal{M}$ will not be able to cash the coin. Thus the responsibility for double-spending each coin is shifted to its witness, who has left a security deposit at $\mathcal{B}$: the main observation here is that merchants are in general always on-line due to the nature of their activity allowing for real-time double-spending check.

$\mathcal{M}$ cashes the coins at $\mathcal{B}$. When presented with payment transcripts, $\mathcal{B}$ verifies that each coin has been signed by the required witness and has not been deposited before. $\mathcal{B}$ then makes a deposit into $\mathcal{M}$'s account, and saves the payment transcripts until the coins become uncashable, in order to detect misbehaving merchants. If a certain payment transcript is signed twice by some witness, $\mathcal{B}$ can punish this witness using the security deposit that was left by $\mathcal{M}$. In particular, the security deposit should cover the double-spent coins out of which the cheated merchants can be paid.

It may happen that a coin is unusable due to the unavailability of its assigned witness. To decrease probability of such event, one can use, say, three witnesses per coin and require any two of them to sign. However, in the event that this still does not help and the coins are unspendable, we assign to each coin C two expiration dates, a "soft" expiration date after which it becomes unspendable, but can still be exchanged for a new coin, and a hard expiration date, after which it becomes completely void. This exchange can be done when buying new coins, by submitting coins past their first expiration date as well. This approach allows clients to renew unused coins and to recover from faulty witnesses.

Witness Motivation and Assignment. Why would a merchant agree to serve as a witness, signing payment transcripts? To see how we can motivate merchants to serve as witnesses, we *first* notice that preventing double-spending helps the community as a whole, since merchants are not left with unpaid transactions due to credit fraud and need not expend extra effort to secure a credit card database: thus we assume that merchants are for the most part cooperative and would in general be willing to do a little extra work to contribute to the health of the community. *Second*, when some merchants still do not see value in doing witness service, the broker can provide incentives to merchants for signing coins, e.g. give discounts on cashing the coins, where the credit given depends on the amount of witness service (e.g. coins signed) the merchant has performed. The merchants that do not sign will pay more fees for cashing coins, while the hardworking witnesses will get sufficient credit to motivate them to continue serving in witness capacity. The exact policy enforced by the broker, though, is beyond the scope of this paper.

Note that in the proposed scheme each coin has a statically assigned witness. The reason why we do not allow witnesses for a coin to change is because, otherwise, a secure witness hand-off would be required when witnesses change. Efficient, secure witness hand-off would have to involve a trusted third party which we wanted to avoid in the design. However, now that the witness assignment is static, we need to figure out how it can be done so that: 1) to maintain untraceability, $\mathcal{B}$ issuing the coin does not know which merchant was assigned to be the witness for this coin, but still 2) $\mathcal{B}$ ensures that the hardworking witness merchants are assigned more coins than others. That is, the client should not be able to skew witness assignment towards (or away from) specific merchants, while $\mathcal{B}$ should not be able to link the coin to specific witness merchants.

Given a secure e-cash protocol with public information, the following scheme provably achieves these aims. Let $h : \{0,1\}^* \rightarrow \{0,1\}^k$ be a cryptographic hash function modelled as a random oracle. Let $\mathcal{W}$ be the current merchants in the network that are participating as witnesses. Based on merchant performance, each merchant $\mathcal{M} \in \mathcal{W}$ is assigned a "witness range", $R_{\mathcal{M}} = [r_{\mathcal{M},1}, r_{\mathcal{M},2}) \subset [0, 2^k)$, such that $R_{\mathcal{M}_1} \bigcap R_{\mathcal{M}_2} = \emptyset, \forall \mathcal{M}_1 \neq \mathcal{M}_2 \in \mathcal{W}$, and $\bigcup_{\mathcal{M} \in \mathcal{W}} R_{\mathcal{M}} = [0, 2^k)$. The merchants that should be assigned more coins will be assigned larger witness ranges. Let us call the unblinded coin together with bro-

ker's signature as bare coin. Given an authentic list of merchants and their witness ranges, the witness for a bare coin could be simply the merchant whose witness range contains $h(\text{bare coin})$. The full-fledged coin C then will be the tuple consisting of the bare coin and the signed witness range assignment of the witness merchant $\mathcal{W}$. Since the bare coin was blinded during the signing, $\mathcal{B}$ will not know the $h(\text{bare coin})$. And since the client cannot forge $\mathcal{B}$'s signature in bare coin, the client will not be able to predict the hash value either.[3] This results in a coin C which contains a (non-malleable) witness assignment, where the witness merchant is randomly selected using the probability distribution imposed by the list of the witness ranges.

Assigned witness ranges may change over time, since merchants may join or leave the network or experience changing ability to sign coins. For that purpose, from time to time, $\mathcal{B}$ may publish a new version of the witness range assignments. We will discuss the specifics of how one attaches a witness to a coin in the later sections.

5. The Protocols

Operations with e-cash involve three protocols: *withdrawal*, in which the client buys coins from $\mathcal{B}$; *payment*, in which the client pays the merchant using these coins; and *deposit*, in which the merchants cash the coins. The interaction between the bank and broker (which can be the same entity) can follow standard financial protocols and is orthogonal to our construction.

Let p and q be two large primes such that $q|p-1$ and $g \in \mathbb{Z}_p^*$ be a random generator of order q. In practice (and the implementation) p will usually be a 1024-bit and q will be a 160-bit prime. Denote by $\langle g \rangle$ the subgroup generated by g and let g_1 and g_2 be two random generators of $\langle g \rangle$.

We assume that it is hard to compute logarithms in $\langle g \rangle$, i.e. given a random generator $\hat{g}$ of $\langle g \rangle$ and $f \in \langle g \rangle$, it is hard to find $a \in \mathbb{Z}_q$ such that $\hat{g}^a = f$. We also choose and fix some public cryptographic hash functions $\mathcal{F} : \{0,1\}^* \rightarrow \langle g \rangle$ and $\mathcal{H} : \{0,1\}^* \rightarrow \mathbb{Z}_q$, which can be easily constructed using standard cryptographic hash functions.

$\mathcal{B}$ chooses a secret key $x \in \mathbb{Z}_q$ and publishes the authenticated key $y = g^x$. The pair (y, x) will be used as a public/private key pair in the *partially blind signature* scheme of Abe and Okamoto [3]: $\mathcal{B}$ signs using x and signatures are verified using the public value y.

Withdrawal Protocol. The withdrawal protocol should have the following essential properties:

1. The (bare) coin, including $\mathcal{B}$'s own signature, should be blinded from $\mathcal{B}$, *i.e.*, $\mathcal{B}$ should not be able to obtain any information about the actual coin other than possibly some agreed-upon public information that is attached to the coin (this includes coin unlinkability as discussed in later sections). Without this property, $\mathcal{B}$ (perhaps in collusion with merchants) might be able to link a coin to a specific user, especially if coins were not purchased anonymously.

2. $\mathcal{B}$ should ensure that the coin is assigned correct expiration dates and witnesses. The only information that $\mathcal{B}$ should know about witness assignment is that the witness is assigned according to the current list of witness ranges. $\mathcal{B}$ should know the exact dates assigned to the coin.

3. Anyone should be able to correctly determine if a given merchant is indeed a witness of a given coin from the coin itself. More precisely, merchants do not need to store the entire history of witness range assignments.

4. Other standard security properties such as strong unforgeability, unexpandibility, unreusability of coins and so on are also required (see Section 6).

Denote by info the explicit information to be added to the coin, and by $I_{\mathcal{M}}$ the unique identifier of merchant $\mathcal{M}$. As mentioned in Section 4, $\mathcal{B}$ publishes when needed a new version of signed witness range assignments $\text{Sig}_{\mathcal{B}}(\text{version/date}, \{I_{\mathcal{M}}, r_{\mathcal{M},1}, r_{\mathcal{M},2}\})$ for each merchant $\mathcal{M} \in \mathcal{W}$, where $[r_{\mathcal{M},1}, r_{\mathcal{M},2})$ is the range assigned to $\mathcal{M}$. The info will include the version/date of the merchant list and two expiration dates (possibly with the denomination of the coin). During coin generation, $\mathcal{B}$ will produce a *partially blind signature* [2] of Abe and Okamoto [3] where info is attached to the coin in non-blinded form. Once the client has unblinded the partially blind signature, thus obtaining the bare coin with the above info and signed by broker $\mathcal{B}$, he/she computes $h = h(\text{bare coin})$ and copies the appropriate $\text{Sig}_{\mathcal{B}}(\text{version/date}, \{I_{\mathcal{M}}, r_{\mathcal{M},1}, r_{\mathcal{M},2}\})$, where $h \in [r_{\mathcal{M},1}, r_{\mathcal{M},2})$ and version/date is the same as in the bare coin, resulting in the full-fledged coin. The full protocol is described in Algorithm 1 and is an adaptation of Abe-Okamoto, where 1) instead of signing an arbitrary msg, $\mathcal{B}$ signs a tuple $(A = g_1^{x_1} g_2^{x_2}, B = g_1^{y_1} g_2^{y_2})$ that will be used during the payment protocol, and 2) we specify the value of info that will be attached to the coin. The values A and B are constructed by the client, who knows the corresponding representation coefficients x_1, x_2, y_1, y_2 with respect to g_1 and g_2. The construction of A and B along with the non-interactive zero-knowledge (NIZK) proofs of representation of A and B are borrowed from Brands [7] and Okamoto [26]. Note that the client can not tamper with bare coin and, at the same time, $\mathcal{B}$ will learn nothing about the bare coin other than the attached info.

As the result of the withdrawal protocol, the client obtains a valid coin $C = (\rho, \omega, \sigma, \delta, \text{info}, A, B, \text{Sig}_{\mathcal{B}}(\text{version/date}, \{I_{\mathcal{M}_C}, r_{\mathcal{M}_C,1}, r_{\mathcal{M}_C,2}\}))$, which contains the signature of $\mathcal{B}$. Anyone can verify validity of the coin by checking validity dates, verifying that the correct witness was assigned to the bare coin and most importantly by verifying $\mathcal{B}$'s signature on the coin by checking that

[3] It is straightforward to prove these properties hold using standard random oracle proof techniques [5]

Algorithm 1 Withdrawal Protocol

0. The client $\mathcal{C}$ and $\mathcal{B}$ agree on the denomination of the coin, the version of the merchant list that will be attached, and on the two expiration dates as explained before. The client pays for the coin using credit card, bank account or through other accepted alternatives. Client can buy several coins at a time (saving on communication cost), but the computation below have to be performed independently for each coin to ensure they are unlinkable.

1. $\mathcal{B} \rightarrow \mathcal{C} : a = g^u, b = g^s z^d$

$\mathcal{B}$ picks random $u, s, d \in \mathbb{Z}_q$ and sends the constructed a, b to the client, where $z = \mathcal{F}(\mathsf{info}) \in \mathbb{Z}_p^*$. The info contains the value of the coin, the version of merchant list, and two expiration dates.

2. $\mathcal{C} \rightarrow \mathcal{B} : e$

The client picks four random values $t_i \in \mathbb{Z}_q, i = 1, ..., 4$ and $x_1, x_2, y_1, y_2 \in \mathbb{Z}_q$, and computes $\alpha = a g^{t_1} y^{t_2}$, $\beta = b g^{t_3} z^{t_4}$, $\epsilon = \mathcal{H}(\alpha||\beta||z||A||B)$ and $e = \epsilon - t_2 - t_4 \mod q$, where $A = g_1^{x_1} g_2^{x_2}$, $B = g_1^{y_1} g_2^{y_2}$. The value of e is sent to $\mathcal{B}$.

3. $\mathcal{B} \rightarrow \mathcal{C} : (r, c, s)$

$\mathcal{B}$ computes $c = e - d \mod q, r = u - cx \mod q$ and sends triple (r, c, s) to the client.

4. The client computes $\rho = r + t_1 \mod q$, $\omega = c + t_2 \mod q$, $\sigma = s + t_3 \mod q$, $\delta = e - c + t_4 \mod q$, and checks equality $\omega + \delta = \mathcal{H}(g^\rho y^\omega||g^\sigma z^\delta||z||A||B) \mod q$. Denote the bare coin $= (\rho, \omega, \sigma, \delta, \mathsf{info}, A, B)$. The client attaches the $\mathsf{Sig}_{\mathcal{B}}(\mathsf{version/date}, \{I_{\mathcal{M}_C}, r_{\mathcal{M}_C,1}, r_{\mathcal{M}_C,2}\})$, where $h(\text{bare coin}) \in [r_{\mathcal{M}_C,1}, r_{\mathcal{M}_C,2})$, resulting in the unblinded coin $C = (\rho, \omega, \sigma, \delta, \mathsf{info}, A, B, \mathsf{Sig}_{\mathcal{B}}(\mathsf{version/date}, \{I_{\mathcal{M}_C}, r_{\mathcal{M}_C,1}, r_{\mathcal{M}_C,2}\}))$.

$\omega + \delta = \mathcal{H}(g^\rho \cdot y^\omega||g^\sigma \cdot \mathcal{F}(\mathsf{info})^\delta||\mathcal{F}(\mathsf{info})||A||B) \mod q$. However, only the coin owner knows the representations of A and B with respect to the tuple (g_1, g_2), which will be used in the payment protocol below. Note that $\mathcal{B}$ does not know which witness was assigned to the coin and the client cannot influence the choice of witness.

Payment Protocol. In the payment protocol, client $\mathcal{C}$ wants to pay for a service provided by merchant $\mathcal{M}$ using coin C. Prior to providing the service, $\mathcal{M}$ will have to determine if the coin has already been spent by contacting the witness of the coin $\mathcal{M}_C$. The protocol must ensure that:

1. If the coin has already been spent, the witness $\mathcal{M}_C$ can provide an unforgeable proof. For more privacy, it is desirable that the proof does not reveal the identity of $\mathcal{M}$ where the coin was previously spent.

2. If $\mathcal{M}$ refuses to provide the service claiming that a coin is being double-spent, $\mathcal{M}$ will be able to convince a third party that the coin was already spent prior to the transaction.

3. Conflict resolution mechanisms such as optimistic fair exchange can be incorporated naturally. The payment transcript should be publicly verifiable and should not reveal secrets of the parties involved. In particular, anyone that sees the transcript should not be able to forge another payment transcript, or cash the coin.

Our payment protocol is similar to the original protocol of Brands. In particular, to pay with the coin, the client will need to provide a non-interactive zero-knowledge proof (NIZK) that it knows the representation of A and B with respect to the tuple (g_1, g_2) inside the coin. The proof will bind the payment transcript to the given merchant and time so that only that merchant will be able to cash the coin. Moreover, given two such payment transcripts, one can extract the secret values x_1, x_2 (and y_1, y_2) of the coin which become the proof that the coin has been double-spent (see the security analysis section for more details). The full protocol is specified in Algorithm 2. Note that since the value d in Step 3 depends not only on the merchant and time but also on the unblinded coin, the client can not spend a coin without knowing the representation of A and B.

Algorithm 2 Payment Protocol

1. $\mathcal{C} \rightarrow \mathcal{M}_C : (coin\_hash, nonce)$

The client contacts the witness of the coin, trying to obtain the commitment that the witness will sign the payment transcript after the transaction. The $coin\_hash$ is computed as $h(\rho, \omega, \sigma, \delta, \mathsf{info}, A, B)$, $I_{\mathcal{M}}$ is the identity of the merchant where the client intends to spend the coin and $nonce = h(salt_{\mathcal{C}}||I_{\mathcal{M}})$, where $salt_{\mathcal{C}}$ is the random value that the client has chosen for this transaction.

2. $\mathcal{M}_C \rightarrow \mathcal{C} : \mathsf{Sig}_{\mathcal{M}_C}(coin\_hash, nonce, h(v), t_e, \mathsf{commit})$

The coin witness provides a signed commitment that it will sign the payment transcript provided that a) the payment transcript (submitted later) will be valid, b) $\mathcal{M}_C$ is actually the witness for the coin, c) the coin was not already spent before the commitment, d) the coin is spent at the merchant encoded in $nonce$, and d) transaction finishes before time t_e. The value v is either some random value (if the coin has not been spent so far), a "salted" payment transcript of this coin or tuple (x_1, x_2) or (y_1, y_2) (if the coin has already been spent). *The witness must not issue new committments on this $coin\_hash$ until this commitment expires.*

3. $\mathcal{C} \rightarrow \mathcal{M} : \text{payment transcript} = (C, r_1, r_2, I_{\mathcal{M}}, date/time), \mathsf{Sig}_{\mathcal{M}_C}(coin\_hash, nonce, h(v), t_e, \mathsf{commit}), salt_{\mathcal{C}}$

The client sends to $\mathcal{M}$ the coin, and $r_1 = x_1 + d \cdot y_1, r_2 = x_2 + d \cdot y_2$, where $d = \mathcal{H}_0(C, I_{\mathcal{M}}, date/time)$. In addition, the commitment from $\mathcal{M}_C$ along with $salt_{\mathcal{C}}$ are sent. $\mathcal{M}$ verifies the broker's signature on the coin (as specified in the discussion of the withdrawal protocol), the correctness of witness assignment, the witness commitment and equality $nonce = h(salt_{\mathcal{C}}||I_{\mathcal{M}})$. In addition, the equality $A \cdot B^d = g_1^{r_1} g_2^{r_2}$ is checked. The merchant rejects if any of the checks fail or if it has already received payment with the same coin.

4. $\mathcal{M} \rightarrow \mathcal{M}_C : \text{payment transcript} = (C, r_1, r_2, I_{\mathcal{M}}, date/time, salt_{\mathcal{C}})$

The payment transcript is sent to the witness (the commitment information is sent only during conflict resolutions), which is verified. The witness will retrieve stored commitment and verify that $nonce = h(salt_{\mathcal{C}}||I_{\mathcal{M}})$, refusing transaction if this check fails. If the coin was spent once prior to the commitment, the witness computes (x_1, x_2) and/or (y_1, y_2) and keeps only this value along with hash of the coin, dropping all transcripts.

5. $\mathcal{M}_C \rightarrow \mathcal{M} : \mathsf{Sig}_{\mathcal{M}_C}(\text{payment transcript})$, or (x_1, x_2) and/or (y_1, y_2), or refusal if $nonce$ did not verify

If $nonce$ did not verify, the witness simply refuses to sign based on that. If the coin is double-spent, the witness sends (x_1, x_2) and/or (y_1, y_2), refusing to sign. Otherwise, it provides the signature on the transcript.

6. $\mathcal{M} \rightarrow \mathcal{C} : \mathsf{service}$, or (x_1, x_2) and/or (y_1, y_2).

The client either obtains the service or is refused with the proof of double-spending.

Note that we shift part of the communication onto the client, which has to obtain a commitment from the witness. Thus, prior to a transaction, the client can be assured that the witness will sign the transcript. The client must construct $nonce$ correctly, for otherwise it will be refused transaction. The commitment has the following properties: 1) the commitment is bound to specific merchant through $nonce$, 2) the witness does not know apriori where the coin will be spent. The witness will sign transcript even if this commitment is used more than once, but in this case the same (faulty) merchant will have to deposit two payment transcripts with the same coin, which will be detected and stopped at the broker.

When a coin is double-spent, the witness does not release information on where the coin was spent before, while it still provides a publicly verifiable proof that the coin has been double-spent. In case of dispute, all transcripts can be given to a third party to decide on further action. In particular, fair exchange protocols may be incorporated into the

transactions. Note that if race conditions exist such that the same coin has been spent at another merchant right after the witness has made the commitment, and therefore the witness was able to generate (x_1, x_2) and/or (y_1, y_2), $\mathcal{M}$ may ask the witness to reveal the committed value v. If the value v does not contain (x_1, x_2) or (y_1, y_2) or a previous payment transcript, this is a proof that the witness violated the protocol. Finally, in case of problems, all communication transcripts can be submitted to a third party for resolution, which can decide who has violated the protocols.

Deposit Protocol. In the deposit protocol, $\mathcal{M}$ submits the payment transcript signed by the witness to $\mathcal{B}$, who verifies the transcript and the witness' signature. Before crediting $\mathcal{M}$'s account, $\mathcal{B}$ also checks if this coin has already been deposited: this is possible if either 1) the same merchant deposits the same coin again or 2) the same witness signed two transcripts for the same coin. In the former case, $\mathcal{M}$ is informed of the mistake and is not credited; in the latter, the witness will be charged for the transaction and $\mathcal{M}$ will be credited from the witness' deposit. The witness can be contacted with the proof (two signatures) that it incorrectly performed its duty, and additional administrative actions (beyond the scope of this paper) can be taken. The protocol is shown in Algorithm 3.

Algorithm 3 Deposit Protocol

1. $\mathcal{M} \rightarrow \mathcal{B}$: payment transcript, $\mathsf{Sig}_{\mathcal{M}_C}$(payment transcript)
$\mathcal{M}$ sends to $\mathcal{B}$ the payment transcript signed by the witness, where payment transcript $= (C, r_1, r_2, I_{\mathcal{M}}, date/time, salt_{\mathcal{C}})$ and the coin $C = (\rho, \omega, \sigma, \delta, \mathsf{info}, A, B, \mathsf{Sig}_{\mathcal{B}}(\mathsf{version/date}, \{I_{\mathcal{M}_C}, r_{\mathcal{M},1}, r_{\mathcal{M},2}\}))$. $\mathcal{B}$ verifies its own signature on the coin, that the coin is still valid and cashable, and that the right witness signed the transcript. Next, $\mathcal{B}$ verifies the signature of the witness on the payment, computes d and checks the equality $A \cdot B^d = g_1^{r_1} g_2^{r_2}$. If at least one test fails, the $\mathcal{B}$ notifies $\mathcal{M}$ of the failure and the protocol ends here.
2. $\mathcal{B}$ searches its database to determine if the bare coin $= (\rho, \omega, \sigma, \delta, \mathsf{info}, A, B)$ has previously been deposited. Two options are possible:
2-a. The coin has *not* been deposited before. In this case, the broker stores the payment transcript along with witness' signature until the coin's second expiration date. $\mathcal{M}$ is credited for the amount of the coin.
2-b. $\mathcal{B}$ finds another deposit of the same coin. If this deposit was made by the same merchant, it will refuse the deposit and inform the merchant of the mistake. If this deposit was made by another merchant, the merchant $\mathcal{M}$ is still credited for the coin amount, but now it is done from the security deposit of the coin's witness $\mathcal{M}_C$. The $\mathcal{M}_C$ is notified appropriately with the proof consisting of two $\mathcal{M}_C$'s signatures on the same coin with different merchants.

Coin Renewal. Each coin has two expiration dates to allow clients to renew unused or unusable coins. After the first date, the coin will no longer be cashable and after the second one it becomes completely void. The coin renewal protocol is described in Algorithm 4. In this protocol, the client submits a coin, which is past the first expiration date but not the second, along with a proof that it knows the representation of A and B inside the coin with respect to g_1 and g_2. Then $\mathcal{B}$ searches its database to find out if the coin has already been cashed or renewed and, if it was, extracts and provides the value of (x_1, x_2) and/or (y_1, y_2) and refuses to renew. Otherwise, the client obtains a new coin similarly to the withdrawal protocol. The protocol is the same as withdrawal with piggy-backed coin verification and can be done when a client buys new coins.

Algorithm 4 Coin Renewal Protocol

0. The client $\mathcal{C}$ and $\mathcal{B}$ agree on the version of the merchant list that will be attached, the value of the new coin, and on the two expiration dates that will be attached to the new coin.
1. $\mathcal{B} \rightarrow \mathcal{C} : a = g^u, b = g^s z^d$
$\mathcal{B}$ picks random $u, s, d \in \mathbb{Z}_q$ and sends the constructed a, b to the client, where $z = \mathcal{F}(\mathsf{info})$. The info contains the value of the coin, the version of merchant list, and two expiration dates.
2. $\mathcal{C} \rightarrow \mathcal{B} : e, C^*, r_1^*, r_2^*$.
The client picks four random values $t_i \in \mathbb{Z}_q, i = 1, ..., 4$ and $x_1, x_2, y_1, y_2 \in \mathbb{Z}_q$, and computes $\alpha = a g^{t_1} y^{t_2}$, $\beta = b g^{t_3} z^{t_4}$, $\epsilon = \mathcal{H}(\alpha||\beta||z||A||B)$ and $e = \epsilon - t_2 - t_4 \mod q$. The client sends e, the old bare coin $C^* = (\rho^*, \omega^*, \sigma^*, \delta^*, \mathsf{info}^*, A^*, B^*)$ (to be renewed) and the proof of knowledge of representation of A^* and B^* for C^*, consisting of r_1^* and r_2^* (where d^* is constructed as in the payment protocol).
3. $\mathcal{B} \rightarrow \mathcal{C} : (r, c, s)$ or (x_1^*, x_2^*) and/or (y_1^*, y_2^*).
$\mathcal{B}$ verifies the correctness of C^*, computes d^* and checks that $A^* \cdot {B^*}^{d^*} = {g_1^*}^{r_1^*} {g_2^*}^{r_2^*}$ holds. $\mathcal{B}$ searches if the coin has been deposited by a merchant or has already been renewed: in this case, $\mathcal{B}$ can compute (x_1^*, x_2^*) and (y_1^*, y_2^*) corresponding to C^* and return them to the client with the refusal to renew. Otherwise, $\mathcal{B}$ computes $c = e - d \mod q$ and $r = u - cs \mod q$, and returns them to the client along with s. The renewal transcript is stored until the C^*'s full expiration.
4. The client computes $\rho = r + t_1 \mod q$, $\omega = c + t_2 \mod q$, $\sigma = s + t_3 \mod q$, $\delta = e - c + t_4 \mod q$, and checks equality $\omega + \delta = \mathcal{H}(g^\rho y^\omega || g^\sigma z^\delta || z || A || B) \mod q$. Denote bare coin $= (\rho, \omega, \sigma, \delta, \mathsf{info}, A, B)$. The client attaches the $\mathsf{Sig}_{\mathcal{B}}(\mathsf{version/date}, \{I_{\mathcal{M}_C}, r_{\mathcal{M},1}, r_{\mathcal{M},2}\})$, where $h(\text{bare coin}) \in [r_{\mathcal{M}_C,1}, r_{\mathcal{M}_C,2})$, resulting in the unblinded coin $C = (\rho, \omega, \sigma, \delta, \mathsf{info}, A, B, \mathsf{Sig}_{\mathcal{B}}(\mathsf{version/date}, \{I_{\mathcal{M}_C}, r_{\mathcal{M}_C,1}, r_{\mathcal{M}_C,2}\}))$.

6. Security

A good review of security requirements for anonymous off-line e-cash systems is given in [35]. However, our requirements slightly differ since our double-spending detection is real-time and coin owners are fully untraceable.

The security of our e-cash system depends on the security of two core components: 1) security of the partially blind signature and 2) security of the representation proof in groups of prime order. We first remark that the partially blind signature of Abe-Okamoto [3] is *strongly unforgeable* which means that the bare coin obtained during withdrawal protocol cannot be altered by the client without invalidating the signature. From the partial blindness of the signature, it follows that the only property about the bare coin that $\mathcal{B}$ learns is the info attached to it: in particular, given two unblinded coins with the same info, $\mathcal{B}$ cannot decide which coin belongs to which instance of withdrawal.

Security of the NIZK proof used in the payment protocol is important as it proves ownership of the coin. During withdrawal, the client chooses random x_1, x_2, y_1, y_2 and constructs $A = g_1^{x_1} g_2^{x_2}$, $B = g_1^{y_1} g_2^{y_2}$: the values of A and B become part of the bare coin and thus can not be altered. The tuples (x_1, x_2) and (y_1, y_2) are called *representations of A and B with respect to generators g_1, g_2*. Finding a second representation of A given one representation, or finding any representation given a random A are both provably equivalent to computing discrete logarithms in $\langle g \rangle$ [7] and

thus assumed to be hard problems. Thus if the client knows a representation of A (B) then we can conclude that 1) the client (perhaps by proxy) actually constructed the coin, and 2) the client knows no other representation of A (B).

The NIZK proof has the following security properties [6, 26]: 1) a client can successfully provide the NIZK proof if and only if he/she knows representations of A and B; 2) if the client submits two successful NIZK proofs with respect to the same coin, then we can extract the representations of A and B from the proofs themselves.[4] Consequently, we can make the following conclusions: 1) only the coin owner can successfully make a payment; 2) seeing a payment transcript does not allow one to generate another payment transcript; 3) if the coin owner double-spends, the representation of A and/or B can be extracted which serves as a definitive proof of double-spending.

On the security of the present scheme, let us first make several basic conclusions based on the previous observations. We note that the bare coin obtained during the withdrawal is non-malleable and contains the version of the list of witness range assignments, so (provided the broker is correct) only one witness can be attached to the coin. Consequently, *the full-fledged coin is non-malleable* as well. Secondly, the bare coin is strongly unforgeable which implies the *unexpandibility* of the coins, i.e., given N coins generated in a valid manner, the attacker should not be able to create $N + 1$ coins.[5] Thirdly, the e-cash scheme satisfies the *unreusability property*, i.e., a coin with an honest witness can be used no more than once at the merchants in the network. Indeed, the coin is non-malleable (including the witness assignment) and can be spent only if the same witness signs different transcripts for this coin. Unless the witness merchant is faulty, the second transcript will allow extraction of a representation of A which can be done only if the coin has been double-spent. In the end, the non-faulty merchant will refuse to sign the second transcript with a definitive proof of double-spending and the coin will be refused. If the witness is faulty and signs two payment transcripts for the same coin, the merchants will still be paid by $\mathcal{B}$ at the expense of the witness. Fourthly, the NIZK ensures that only the coin owner can successfully make a payment to a merchant, resulting in non-malleable payment transcripts.

Now let us see why a coin cannot be traced back to its owner. First, as was mentioned above, when the client spends the coin at $\mathcal{M}$ and $\mathcal{B}$ sees the payment transcript, $\mathcal{B}$ will not be able to tell which client bought this coin among all clients who bought coins with the same info. However, this property is not enough for anonymity if the client buys several coins at a time. More precisely, it is conceivable that the broker could in some way skew the protocol so that two coins withdrawn at the same time may be linkable; in this scenario a broker who cooperates with one merchant may be able to trace a coin spent at another merchant.

To show that such attacks are virtually impossible, or equivalently prove the *unlinkability* property of our e-cash scheme, consider two honest clients $\mathcal{C}_1$ and $\mathcal{C}_2$ who engage with $\mathcal{B}$ in withdrawal protocols with the same info such that the first client generates two unblinded coins C_1, C_2, and the second client obtains the unblinded coin C_3. Now, suppose that the clients give the coins (and all the secret information) to the challenger who plays the following game with the adversarial broker:

– The challenger gives C_1, along with all associated secret information, to $\mathcal{B}$. $\mathcal{B}$ knows this coin was generated by $\mathcal{C}_1$.

– The challenger at random $i \in \{2, 3\}$ and then gives C_i to $\mathcal{B}$ along with secret information about the coin.

– $\mathcal{B}$ guesses i and wins if its answer is correct.

If $\mathcal{B}$ can somehow link C_1 and C_2 together, then it will be able to win with probability non-negligibly better than simple guessing. More precisely, let us say that the e-cash provides *coin unlinkability* if in the above game the adversary cannot guess correctly with probability $1/2 + \delta$ for non-negligible $\delta > 0$. Coin unlinkability in this sense is essentially a direct consequence of the blindness property. Indeed, according to the blindness property of the coins, given C_2 and C_3 in a random order (together with the secret information about the coins), $\mathcal{B}$ will not be able to decide which coin is C_2 and which one is C_3 other than with negligible advantage. This readily implies coin unlinkability.

Besides the above security properties, in case of a conflict, all transcripts and commitments can be given to a trusted third party for arbitration. It is a routine exercise to verify that the third party will be able to effectively determine the violator of the protocols, and is left to the reader.

7. Efficiency and Implementability

Implementation. Our implementation consists of four components, totalling approximately 1200 lines of code (LOC): a broker server (158 LOC), merchant server (158 LOC), witness server (294 LOC), and client (258 LOC). The witness and merchant servers are designed to be run at the same time on the same physical hardware, but not in the same memory space (for increased security).

We chose the Python scripting language to implement all four components due to Python's ease in handling web services and distributed applications, as well as the availability

[4] If we have $r_1, r_2, r_1', r_2', d \neq d'$ such that $A \cdot B^d = g_1^{r_1} g_2^{r_2}$ and $A \cdot B^{d'} = g_1^{r_1'} g_2^{r_2'}$, then we obtain $B = g_1^{(r_1'-r_1)/(d'-d)} g_2^{(r_2'-r_2)/(d'-d)}$ and thus the tuple $((r_1' - r_1)/(d' - d), (r_2' - r_2)/(d' - d))$ is a representation of B. Then a representation of A can be easily obtained from $A \cdot B^d = g_1^{r_1} g_2^{r_2}$ and knowledge of d, r_1, r_2.

[5] Indeed, consider the game in which a client attempts to generate, for info of his choice, more valid signatures than it requested from the signing oracle for this value of info. According to Abe-Okamoto, the client has only a negligible advantage in this game, provided that the number of requested signatures for any fixed info is polynomial in the logarithm of the security parameter. This readily implies the *unexpandibility* property of e-cash.

of unbounded-width integers and easy-to-use cryptographic libraries. The broker, merchant, and witness components are implemented as stand-alone web service providers, but can easily be changed into drop-in modules for existing web servers (such as Java application servers or Apache).

We use a (mostly) stateless transaction design for our web servers, based largely on REST principles [16]. We keep minimal session state at broker, merchant, and witness, and rely on the client to transmit all state information when requesting transactions. For state information that was originally generated by someone other than the client, the transaction request contains a signature on the externally-generated information to prevent modification by the client/intermediary. All state is encoded as universal resource identifiers (URIs) and transferred along with the transaction request. This design trades implementation simplicity for increased communication overhead, as state must be transferred repeatedly throughout a single logical transaction (a logical transaction may include multiple communication sessions between several servers). If state needs to be kept secret, an encrypted, timestamped, and signed blob can be transmitted as a state identifier. Alternatively, the system could keep state and use transaction identifiers.

Table 1. Number of cryptographic operations

		Exp	Hash	Sig	Ver
Withdrawal	Client	12	4	0	1
	Broker	3	1	0	0
Payment	Client	0	3	0	1
	Witness	7	6	2	1
	Merchant	7	6	0	3
Deposit	Merchant	0	0	0	0
	Broker	6	4	0	1
Coin Renewal	Client	12	5	0	1
	Broker	9	4	0	0

Complexity analysis. Since we are using URL-encoded data transfer, all state information is encoded as text when transferred over the network. This imposes higher communication overhead than binary file transfer, but compression and/or base64 data encoding can be used if greater communication efficiency is required. Furthermore, there is a trade-off between how much state is kept by the servers and how much information has to be transferred for each request. For ease of implementation we chose to keep as little state is possible at broker and merchant/witness servers, but we can decrease communication overhead at the cost of more complicated server logic by offloading more state off the client onto the servers.

The number of cryptographic operations for each protocol is listed in Table 1, where we look at a typical scenario without double-spending incidents. In case of double-spending by the client at a different merchant (than the one where the coin was spent originally), the communication overhead stays the same while 1) the merchant will have to do 2 additional exponentiations, but one signature verification less, and 2) the witness will either be spared all significant crypto operations (returning previously computed x_1/x_2) or will have to do only two exponentiations. If the witness signed two transcripts for the same coin, spent at different merchants, the broker will detect this when the second payment transcript is deposited (note that both merchants will be paid, with the second payment coming from the witness' security deposit) – in this case, broker's computational overhead does not change; however, the broker will contact this witness (perhaps in off-line manner) to resolve the issue. If the same merchant attempts to deposit the same coin again (even with different witness' signatures), the merchant will not be paid and the computational overhead of the broker remains the same.

The withdrawal and renewal protocols each require two rounds of message exchange between the broker and client, and payment requires 3 rounds of message exchange (2 for payment, and 1 for commitment). The deposit protocol is one-sided, only requiring the merchant to send one message to the broker. Note that total computational complexity per transaction in terms of real-time will be significantly less than communication overhead: round-trip time on WAN is expected to be at least 50-100 ms (observed on PlanetLab nodes in the US), while the aggregated computational complexity per transaction is expected to be 30 ms or less when implemented in OpenSSL (on a P4 3.2 GHz desktop).

Experimental results. To determine the viability of our protocol for real-world deployment, we measured the time and bandwidth for our payment protocol, and compared the results to the time required to download and render the advertisements on a popular Internet website.

Table 2. Wall-clock runtime and bandwidth for payment protocol over 100 trials

	Client total time	Client bytes transmitted
Average	1789ms	1.6KB
St. dev.	324ms	1.3B

To determine the overall performance of the payment protocol, we simulated 100 runs of the protocol using three randomly selected PlanetLab nodes in diverse geographic locations across the United States.[6] The results of these trials can be found in table 2. The average time over 100 runs to complete the payment protocol, including contacting the witness, was 1.8s. We note that this is a worst-case figure, as the client can obtain a witness commitment on the coin at any time between when the coin is withdrawn from the broker, and the time the coin is spent; several optimizations could further reduce the computation time as well.[7]

For comparison purposes, an informal survey of a popu-

[6] The client and broker were located in Wisconsin, the witness in California, and the merchant in Massachusetts.

[7] The primary source of overhead is in Python's native bignum and crypto libraries, e.g. the average wall-clock time for an RSA signature is 250ms, compared to 4.8ms using OpenSSL.

lar ad-supported web site[8] shows that it serves up 37.13KB in two ad images and associated links for the main page. The total transfer overhead for the client in our protocol is around 1.6KB, with merchant and witness overheads on the order of 4KB. So, our protocol is more efficient than advertisement image-based payment from a network utilization standpoint. Using the same web site, we performed a number of load and render timing tests, and found that it takes on average 0.9 seconds to fetch the website and render a text-only version, ignoring images and scripts. Since we do not load images nor process scripts (which may load additional images or content), this represents the low end estimate, but gives a good benchmark as to what end-users may expect in terms of web page load times.

We can conclude that our protocol is viable in real-world commercial environments with Internet-like communication latencies. Communication overhead itself (the amount of data transferred) can be reduced using known compression techniques, by increasing statefulness of all parties, or a combination of both.

8. Conclusion and Future Work

In this paper, we have proposed a framework for anonymous electronic cash that prevents double-spending without an online trusted authority or special-purpose hardware. Our scheme leverages the power of peer-to-peer systems to provably and efficiently prevent double-spending while retaining conceptual simplicity. Conceptually, the scheme replaces trusted hardware observers as in Brands' protocol with a large group of mostly trustworthy hosts. We have demonstrated the simplicity of our approach with a prototype implementation, and reported on experiments that confirm its efficiency. Our experimental results and simple complexity analysis suggest that the scheme could easily handle web-based mini-payments for many merchants. In addition, The accompanying cryptographic protocols can easily be extended to provide additional functionalities such as escrow service.

As far as incentives, our scheme shifts responsibility for double-spending prevention from the users, who do not benefit from it, to the merchants, who do. As a result, if the coins belonging to some client are stolen, the damage to the client will consist only of the value of the stolen coins. We believe that this more closely aligns the interests and security obligations of the parties involved than the current credit card infrastructure.

The scheme proposed in this paper does not allow for aggregation of transactions, which makes it less efficient than desired. Thus it will be interesting to investigate how coin withdrawals, payments and other protocols can be aggregated together without loss of security. Moreover, an interesting question for further work is how our protocols can be modified to accommodate additional notions of e-cash, such as divisible and unlinkable e-cash.

[8] CNN.com

Acknowledgements. This work was partially supported by NSF grants CNS-0546162 and CNS-0448423.

References

[1] Ukash. Smart Voucher Ltd. `http://www.ukash.com`.

[2] M. Abe and E. Fujisaki. How to Date Blind Signatures. In *ASIACRYPT*, 1996.

[3] M. Abe and T. Okamoto. Provably Secure Partially Blind Signatures. In *CRYPTO*, 2000.

[4] R. Anderson, C. Manifavas, and C. Sutherland. Netcard - A Practical Electronic Cash System. In *Fourth Cambridge Workshop on Security Protocols*, 1996.

[5] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS*, 1993.

[6] S. Brands. An Efficient Off-line Electronic Cash System Based On The Representation Problem. Technical Report CS-R9323, CWI, 1993.

[7] S. Brands. Untraceable Off-line Cash in Wallets with Observers. In *CRYPTO*, 1993.

[8] J. Camenisch, S. Hohenberger, and A. Lysyanskaya. Compact e-cash. In *EUROCRYPT*, 2005.

[9] A. Chan, Y. Frankel, P. MacKenzie, and Y. Tsiounis. Misrepresentation of Identities in E-cash Schemes and How To Prevent It. In *ASIACRYPT*, 1996.

[10] A. Chan, Y. Frankel, and Y. Tsiounis. How to Break and Repair E-cash Protocols Based on the Representation Problem. NU-CCS-96-05, Northeastern Univ., 1996.

[11] D. Chaum. Blind Signatures for Untraceable Payments. In *CRYPTO*, 1982.

[12] D. Chaum, A. Fiat, and M. Naor. Untraceable Electronic Cash. In *CRYPTO*, 1988.

[13] D. Chaum and T. B. Pedersen. Wallet databases with observers. In *CRYPTO*, 1992.

[14] D. Chaum and T. P. Pedersen. Transferred cash grows in size. In *EUROCRYPT*, 1992.

[15] T. Eng and T. Okamoto. Single-term divisible electronic coins. In *EUROCRYPT*, 1994.

[16] R. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, 2000.

[17] Y. Frankel, Y. Tsiounis, and M. Yung. Indirect discourse proofs achieving fair off-line ecash. In *ASIACRYPT*, 1996.

[18] R. Hauser, M. Steiner, and M. Waidner. Micro-payments based on iKP. TR 2791 (#89269), 1996.

[19] J.-H. Hoepman. Distributed Double Spending Prevention. Manuscript, 2006.

[20] M. Jakobsson, D. MRaihi, Y. Tsiounis, and M. Yung. Electronic Payments: Where Do We Go from Here? In *CQRE*, 1999.

[21] S. Jarecki and A. Odlyzko. An Efficient Micropayment System Based on Probabilistic Polling. In *Financial Cryptography*, 1997.

[22] C. Jutla and M. Yung. PayTree: "Amortized-Signature" for Flexible MicroPayments. In *Electronic Commerce*, 1996.

[23] R. J. Lipton and R. Ostrovsky. Micro-Payments via Efficient Coin-Flipping. In *Financial Cryptography*, 1998.

[24] M. Manasse. The Millicent protocols for electronic commerce. Manuscript, 1995.
[25] S. Micali and R. L. Rivest. Micropayments revisited. In *CT-RSA*, 2002.
[26] T. Okamoto. Provably Secure and Practical Identification Schemes and Corresponding Signature Schemes. In *CRYPTO*, 1992.
[27] T. Okamoto. An efficient divisible electronic cash scheme. In *CRYPTO*, 1995.
[28] T. Okamoto and K. Ohta. Universal electronic cash. In *CRYPTO*, 1991.
[29] I. Osipkov, P. Wang, N. Hopper, and Y. Kim. Robust Accounting in Decentralized P2P Storage Systems. In *ICDCS*, 2006.
[30] J. C. Pailles. New protocols for electronic money. In *AUSICRYPT*, 1992.
[31] T. P. Pedersen. Electronic Payments of Small Amounts. Technical Report DAIMI PB-495, Aarhus University, 1995.
[32] R. L. Rivest. Electronic Lottery Tickets as Micropayments. In *Financial Cryptography*, 1997.
[33] R. L. Rivest and A. Shamir. PayWord and MicroMint: Two Simple Micropayment Schemes. In *Security Protocols*, 1996.
[34] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Peer-to-Peer Lookup Service for Internet Applications. In *ACM SIGCOMM*, 2001.
[35] Y. S. Tsiounis. *Efficient Electronic Cash: New Notions and Techniques*. PhD thesis, 1997.
[36] N. van Someren, A. Odlyzko, R. Rivest, T. Jones, and D. Goldie-Scot. Does Anyone Really Need MicroPayments? In *Financial Cryptography*, 2003.
[37] K. Wei, A. J. Smith, Y.-F. R. Chen, and B. Vo. WhoPay: A Scalable and Anonymous Payment System for Peer-to-Peer Environments. In *ICDCS*, 2006.
[38] D. Wheeler. Transactions Using Bets. In *Security Protocols*, 1996.
[39] S. Xu, M. Yung, G. Zhang, and H. Zhu. Money Conservation via Atomicity in Fair Off-Line E-Cash. In *Information Security Workshop*, 1999.
[40] B. Yang and H. Garcia-Molina. PPay: Micropayments for Peer-to-Peer Systems. In *CCS*, 2003.