

WhoPay: A Scalable and Anonymous Payment System for Peer-to-Peer Environments

Kai Wei

CS Division, Dept. of EECS
University of California, Berkeley
Berkeley, CA 94720 USA
kwei@cs.berkeley.edu

Yih-Farn Robin Chen

AT&T Labs-Research
180 Park Ave
Florham Park, NJ 07932-0971 USA
chen@research.att.com

Alan J. Smith

CS Division, Dept. of EECS
University of California, Berkeley
Berkeley, CA 94720 USA
smith@cs.berkeley.edu

Binh Vo

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02138 USA
bdv@mit.edu

Abstract

An electronic payment system ideally should provide security, anonymity, fairness, transferability and scalability. Existing payment schemes often lack either anonymity or scalability. In this paper we propose WhoPay, a peer-to-peer payment system that provides all the above properties. For anonymity, we represent coins with public keys; for scalability, we distribute coin transfer load across all peers, rather than rely on a central entity such as the broker. This basic version of WhoPay is as secure and scalable as existing peer-to-peer payment schemes, while providing a much higher level of user anonymity. We also introduce the idea of real-time double spending detection by making use of distributed hash tables (DHT). Simulation results show that the majority of the system load is handled by the peers under typical peer availability, indicating that WhoPay should scale well.¹

1 Introduction

E-commerce is rapidly becoming the preferred way for many consumers to obtain goods and services. Payments for such transactions on the Web are frequently fulfilled using credit cards or an online payment system such as Pay-

Pal [13]. These electronic payment systems generally incur considerable cost per transaction. For example, most credit card processors charge merchants a minimum fee of between 15 and 35 cents per transaction [10]. As a consequence, these payment schemes are generally considered unsuitable for items that cost \$5 or less. Micropayment systems, which try to aggregate many small micropayments into a few bigger payments, are designed to address this issue.

Another issue with traditional payment technologies like the credit card system and PayPal is the lack of privacy provided to the parties involved in the transactions. With credit cards or PayPal, the identities of the payer and payee of each transaction are exposed not only to each other, but also to the credit card companies or PayPal.com. These exposed identities, together with the transaction itself, can reveal precious or sensitive information about the parties involved. In response to this concern, numerous anonymous payment systems [25] have been proposed to hide user identities during transactions, mostly by using blind signatures [6] or public key cryptography.

Unfortunately, total anonymity makes it more difficult to punish fraud such as double spending and enables crimes such as blackmail and money laundering. What we really want is a payment system where users remain anonymous under normal circumstances but a trusted authority, called the judge, can act on behalf of law enforcement to reveal relevant identities when appropriate. The notion of fairness was introduced by Camenisch [3] to describe this property. Vo and Hohenberger have proposed such a fair system [25].

¹Principal funding for this research has been provided by AT&T Laboratories; the work conducted at UC Berkeley has also been supported by the State of California under the MICRO program and by Toshiba Corporation.

A common characteristic of all the above payment schemes is the use of a central authority, which we refer to in general as the broker. This presents a scalability and performance bottleneck. While credit card companies and PayPal have so far been able to sustain the ever growing transaction load by increasing investment in hardware, this certainly sets the threshold for entry to the payment business very high and makes it infeasible for use in many applications. For example, one can imagine a pay-per-download file sharing system, where a virtual payment system is used to encourage fair sharing of resources among peers. Relying on a broker in such a system would present an unfair cost to a single entity.

PPay [26] is a scalable payment system that is inspired by the success of P2P file sharing systems. Such systems as KaZaA [12] can scale to millions of peers because they pool together and harness the massive resources at the “edge” of the network, rather than relying on expensive centralized resources. As noted by its authors, PPay exploits two main characteristics of P2P applications:

- Peers are generally both consumers and merchants. Thus, coins can be respend repeatedly between users before involving the broker.
- Redistributing the broker’s load onto the peers grants better scalability and performance properties than in existing payment systems.

PPay is secure, fair and scalable, but provides no anonymity. In contrast, the Vo-Hohenberger system is secure, anonymous and fair, but is not scalable. In this paper, we propose WhoPay, a P2P payment system that is secure, anonymous, fair, and scalable, thus combining the best of both worlds.

The rest of the paper is organized as follows. In the next section, we will formally define our design goals. Then in Section 3, we will describe some background information that should help us understand WhoPay’s architecture. This is followed by the description of the basic version of WhoPay in Section 4. Several extensions to this basic design will be introduced in Section 5, including a real-time double spending detection mechanism and others that further improve the anonymity property of WhoPay. We present our simulation work in Section 6 and discuss related work in Section 7. We conclude in Section 8.

Notation: We will let B denote the broker, pk_X the public key of some entity X , sk_X the private key of X , and gk_X the group private key of X . A message M signed by some key K is denoted as $\{M\}_K$.

2 Design Goals

Vo and Hohenberger defined a set of desirable properties for digital payment systems, denoted SAFT, which

stands for Security, Anonymity, Fairness, and Transferability [25]. These properties were defined with the assumption that every coin transfer goes through the broker. We will adopt their terminology, but slightly redefine each property to make it applicable to our peer-to-peer design.

- *Security:* The value of coins can not be tampered with. This means, only the broker can generate coins or increase the value of coins, and only the current holder of a coin can transfer, destroy, or decrease the value of the coin. This guarantees that no user can manipulate the system for profit or to harm another.
- *Anonymity:* Payer and payee do not need to reveal their identities to any third party. This means, without the help of the judge, nobody (other than the participants themselves) can identify the participants of a transaction with probability better than random guessing. Optionally, payer and payee can hide identities from each other.
- *Fairness:* The broker and the judge, working together, can reveal the identities of all parties involved in a particular transaction. If possible, this process should not reveal any information about other transactions.
- *Transferability:* The recipient of a coin can use the same coin to pay another user without identifying himself to the broker. All systems mentioned in this paper support transferability and thus this property will not be the focus of our discussion.

Additionally, we want to reduce broker load:

- *Scalability:* The load of any particular entity does not grow to be unmanageable as the size of the system increases. In particular, the majority of the transaction load should be distributed among peers rather than handled by the broker.

3 Preliminaries

Before we present the WhoPay design, we first briefly describe some background information that will help us understand the architecture of WhoPay, namely PPay and group signatures.

3.1 PPay

PPay is a payment scheme designed for P2P systems. In PPay, user U purchases coins from the broker, and hence becomes the *owner* and *holder* of the coins. To spend the coins he owns, U issues the coins to another user, say V . After the issue, V becomes the current *holder* of the coins, but U remains their owner. If V wants to pay yet another

user W with these coins, he can *transfer* these coins to W via U , the coins' owner. After the transfer, V relinquishes his holdship of the coins and W becomes the current holder of the coins; U remains the owner of the coins. W can further transfer these coins to others, and so on. Only the holder of a coin can spend the coin. Or, the holder can choose to deposit the coin at the broker for cash.

The main challenge in this scheme is to ensure that the security properties are not compromised, since we now want high frequency operations such as transfers to be performed by untrusted peers. As PPay is designed as a micropayment scheme, in that each payment is of a small amount, utmost security is not required; a security model where fraud is detectable (even after the fact) and punishable is probably good enough in most cases. PPay achieves such security as follows.

When user U purchases a coin from the broker, the coin is in the following form:

$$C = \{U, sn\}_{sk_B}$$

where sn is the serial number of a coin that uniquely identifies it. Once issued, the coin becomes:

$$Coin = \{C, H, seq\}_{sk_U}$$

where H is the current holder of the coin, and seq is a sequence number. The coin owner maintains a sequence number counter for the coin and increments the coin's sequence number each time it is issued or transferred. For example, to issue the coin C to user V , U sends V :

$$C_V = \{C, V, seq_1\}_{sk_U}$$

which also serves as a proof of issue. Now for V to transfer the coin to W , V sends the following transfer request to owner U :

$$\{W, C_V\}_{sk_V}$$

and U will keep a record of this transfer request in order to later prove that V has "relinquished" the holdship of the coin, in case of a dispute. Finally, U sends W :

$$C_W = \{C, W, seq_2\}_{sk_U}$$

which also serves as a proof of transfer. Note that seq_2 must be greater than seq_1 .

In summary, a coin explicitly contains the identities of both owner and holder; users sign their messages with their private keys, and keep audit trails of these signatures. These features ensure the "good enough" security mentioned earlier.

Finally, in practice peers come and go, so how do we deal with coins whose owners are offline (we will call such coins "offline coins" from now on)? To address this issue, PPay includes a downtime protocol, in which the broker

temporarily handles the transfer/renewal of offline coins and keeps relevant state. Peers must synchronize state with the broker after they rejoin the system.

It is easy to see that PPay provides very weak, if any, anonymity for the parties involved in a transaction. For example, during coin transfer, the payee knows who the payer is and vice versa, and the coin owner knows who the payee and the payer are and thus can construct a complete transaction history for each coin it owns. In Section 4, we will describe how to modify this scheme to provide anonymity while preserving security, fairness and scalability.

3.2 Group Signatures

In the group signature protocol proposed by Chaum et al [7], a group consists of n private keys G_1, \dots, G_n , one master public key G_p , and one master private key G_s . Each of G_1, \dots, G_n can be used to sign a message. The master public key G_p can be used to verify that the message was signed by one of G_1, \dots, G_n , but cannot tell by which one. The master private key G_s can be used to pinpoint which key was used. G_s is also used to generate new private keys.

WhoPay uses group signatures to achieve fairness. Every user is required to register with a trusted authority, called the *judge*. The judge assigns each user U a (distinct) private key, denoted as gk_U , from a group² and records the user's identity with the private key. The judge also keeps the master private key to herself. (In practice, this master private key can be divided among N judges using Shamir's secret sharing protocol [22] and at least K judges are needed in order to recover the key; but we will make this assumption implicit in the rest of our discussions.) Whenever a user wants to remain anonymous, it signs its messages with its group private key rather than its regular private key. These signatures allow everyone to verify (using the master public key) that the signer is a legitimate user in the system but do not expose its true identity under normal circumstances. However, once a fraud is detected, the judge can be called in to reveal the identities of the bad guys. This way, anonymity is preserved and justice is served.

4 WhoPay

4.1 Overview

WhoPay inherits its basic architecture from PPay. Coins have the same lifecycle as in PPay. Users purchase coins from the broker and spend them by issuing them to other users, who can either spend them by transferring them or deposit them at the broker for cash. Coins must be renewed periodically to retain their value. Coins get trans-

²Yes, all users belong to the same group in WhoPay.

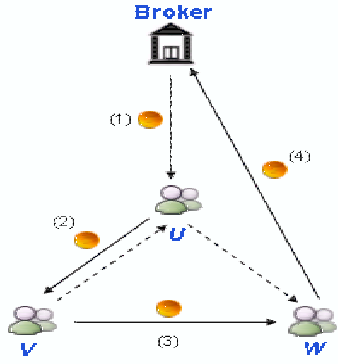


Figure 1. The WhoPay Model: (1) U purchases coin from broker; (2) U issues coin to V ; (3) V transfers coin to W through U ; (4) W deposits coin at broker.

ferred/renewed via the coins' owners if they are online, or via the broker otherwise (Figure 1).

The first major difference of WhoPay from PPay is that coins are identified by public keys, rather than serial numbers. To purchase a coin, user U generates a random public/private key pair pk_{C_U}/sk_{C_U} , keeps the private key sk_{C_U} secret and asks the broker to sign the public key pk_{C_U} .³ The broker sends the coin back in the following form:

$$C = \{U, pk_{C_U}\}_{sk_B}$$

where pk_{C_U} should (with very high probability) uniquely identify the coin. Once issued, the coin becomes:

$$Coin = \{C, pk_{C_H}, seq, exp_date\}_{sk_U}$$

where H is the current holder of the coin, exp_date is the expiration date of the coin, and seq is the sequence number of the coin and serves the same function as in PPay. For example, to issue the coin C , the payee V generates a random public/private key pair pk_{C_V}/sk_{C_V} , keeps the private key sk_{C_V} secret and asks the coin owner U to sign the binding (pk_{C_U}, pk_{C_V}) . The binding (pk_{C_U}, pk_{C_V}) means "coin pk_{C_U} is now represented by pk_{C_V} ", and is a way of conveying the information of who the current holder of a coin is, in that whoever knows the private key sk_{C_V} is the current holder of the coin pk_{C_U} . At any point of time, each user remembers one such binding for each coin it owns. To complete the issue procedure, U sends V :

$$C_V = \{C, pk_{C_V}, seq_1, exp_date_1\}_{sk_U}$$

³As different users generate these public/private key pairs independently, there is a probability of key collision. Key size and renewal period can be chosen to keep this probability small enough so that, the broker can absorb the risk.

which also serves as a proof of issue. Similarly, for V to transfer the coin, the intended payee, say W , also generates a random public/private key pair pk_{C_W}/sk_{C_W} , keeps the private key sk_{C_W} secret and sends the public key pk_{C_W} to V . V then sends the following transfer request to owner U :

$$\{\{pk_{C_W}, C_V\}_{sk_{C_V}}\}_{gk_V}$$

and U will keep a record of this transfer request in order to later prove that V has "relinquished" the holdership of the coin, in case of a dispute. Finally, U sends W :

$$C_W = \{C, pk_{C_W}, seq_2, exp_date_2\}_{sk_U}$$

which also serves as a proof of transfer. Note that seq_2 must be greater than seq_1 .

4.2 Protocol Details

The details of the WhoPay protocols are given below.

Purchase: To purchase a coin from the broker, user U generates a random public/private key pair pk_{C_U} and sk_{C_U} . He keeps sk_{C_U} to himself and sends pk_{C_U} along with his identity (e.g., in the form of a public key certificate) signed by his private key sk_U to the broker. After verifying the signature, the broker adds pk_{C_U} to the list of valid coins, signs the coin with its private key and sends it back to U . The transaction completes after U verifies the broker's signature. It should be straightforward to modify this procedure to purchase coins in batch.

Issue: For U to issue V a coin pk_{C_U} , V generates a random public/private key pair pk_{C_V} and sk_{C_V} , keeps sk_{C_V} to himself, and sends pk_{C_V} to U . U sends V the broker-signed coin pk_{C_U} , and answers a challenge by V to prove he is the owner of the coin. U then updates its coin binding list to bind pk_{C_U} to pk_{C_V} , a randomly chosen sequence number and an appropriate expiration date. U signs the binding with sk_{C_U} , and sends V the signed binding, which serves as a proof of issue of the coin to V . The transaction completes after V verifies the signature.

Transfer: For V to transfer W a coin pk_{C_U} , W generates a random public/private key pair pk_{C_W} and sk_{C_W} , keeps sk_{C_W} to himself, and sends pk_{C_W} to V . V sends the coin owner U a transfer request identifying pk_{C_U} and pk_{C_W} . The transfer request is signed with both sk_{C_V} and V 's group private key gk_V , with the first to prove V 's holdership of the coin and the second to help ensure the fairness of the system. After receiving this transfer request and verifying it is a valid request, U sends W the broker-signed coin pk_{C_U} , and answers a challenge by W to prove he is the owner of the coin. U then updates its coin binding list to bind pk_{C_U} to pk_{C_W} , an incremented sequence number and an appropriate new expiration date. U signs this binding with sk_{C_U} and sends W the signed binding, which serves

as a proof of transfer of the coin to W . The transaction completes after W verifies the signature.

Deposit: For W to deposit a coin pk_{C_U} , it sends a deposit request to the broker identifying the coin to be deposited. The deposit request is signed with both sk_{C_W} and W 's group private key gk_W . After receiving this deposit request and verifying it is a valid request, the broker sends payment to W .

Renewal: For W to renew a coin pk_{C_U} , it sends a renewal request to the coin owner U identifying the coin to be renewed. The renewal request is signed with both sk_{C_W} and W 's group private key gk_W . After receiving this renewal request and verifying its validity, U updates its binding for pk_{C_U} with an incremented sequence number and an appropriate new expiration date. U signs this updated binding with sk_{C_U} and sends W the signed binding, which serves as a proof of renewal. The transaction completes after W verifies the signature.

Downtime transfer: For V to transfer W a coin pk_{C_U} (when the coin owner U is down) via the broker, W generates a random public/private key pair pk_{C_W} and sk_{C_W} , keeps sk_{C_W} to himself, and sends pk_{C_W} to V . V sends the broker a transfer request identifying pk_{C_U} and pk_{C_W} . The transfer request is signed with both sk_{C_V} and V 's group private key gk_V . After receiving this transfer request and verifying it is a valid request, the broker records the binding of pk_{C_U} to pk_{C_W} , an incremented sequence number and an appropriate new expiration date. The broker signs this binding with sk_B and sends W the signed binding, which serves as a proof of transfer of the coin to W . The transaction completes after W verifies the signature.

In the details, there are two flavors of the downtime transfer protocol, depending on whether the coin was last issued/transferred/renewed through its owner, or was it transferred/renewed through the broker. In the first case, most likely the broker has not established any state about the coin and thus needs to verify the coin owner's signature. In the second case, most likely the broker has the up-to-date binding information for the coin and only needs to perform a bit-by-bit comparison of the signed binding received to the locally stored signature.

Downtime renewal: For V to renew a coin pk_{C_U} via the broker, V sends the broker a renewal request identifying the coin to be renewed. The renewal request is signed with both sk_{C_V} and V 's group private key gk_V , with the first to prove V 's holdership of the coin and the second to help ensure the fairness of the system. After receiving this renewal request and verifying it is a valid request, the broker records the binding of pk_{C_U} to pk_{C_V} , an incremented sequence number and an appropriate new expiration date. The broker signs this binding with sk_B and sends V the signed binding, which serves as a proof of renewal of the coin. The transaction completes after V verifies the signature. Simi-

lar to the downtime transfer case, there are two flavors of the downtime renewal protocol, depending on whether the coin was last issued/transferred/renewed through its owner, or was it transferred/renewed through the broker.

Sync: For U to synchronize state with the broker after it rejoins the system, it identifies itself to the broker and proves its claimed identity through a challenge-response procedure. The broker then looks up the bindings for the coins whose owner is U , which it has been maintaining for U during U 's downtime, signs them with its private key sk_B , and sends it to U . After verifying the signed bindings, U updates its coin binding list accordingly.

In summary, coin ownership is still exposed as in PPay, but coin holdership is hidden⁴. Peers only use their private keys to sign messages when they play the role of coin owners, e.g., when they issue coins or handle coin transfers/renewals. When peers act as coin holders, e.g., when they transfer or deposit coins, they use two keys to sign their messages. The first is the coin private key that proves the peer's holdership of the coin and the other is the peer's group private key. Neither signature reveals the peer's identity during normal operations and the group signature allows the identity to be recovered by the judge in exception cases, e.g., in order to identify culprits when fraud is detected.

4.3 System Properties

In this section, we analyze the properties of WhoPay to evaluate how well our design goals outlined in Section 2 have been met.

Security. WhoPay can protect against any attack on coin value that PPay can. As described earlier, WhoPay inherits its basic architecture from PPay, and extends it by replacing coin serial numbers with public/private key pairs and using group signatures to provide fairness. Thus, for an instance of a PPay system, it is trivial to construct a parallel instance of a WhoPay system by generating group keys for the appropriate entities and public encryption keys for all coins in the system. If an adversary could cheat the broker by creating coins in such a WhoPay system, he would be able to do so in a PPay system as well, since he could simply remove the generated encryption keys and replace them with an appropriate serial number. If an adversary could cheat a user by invalidating his coins, those coins would also be invalidated in a corresponding PPay system.

Furthermore, the security for coin holders is made stronger by the use of public key encryption. Since any transfer of a coin requires a signature by the private key of the coin holder which is checked by the coin owner, no adversary can force a holder to relinquish a coin online without colluding with the coin owner under the strong RSA

⁴In section 5, we will show how to anonymize coin ownership as well.

assumption. Even if an adversary includes collusion with the owner, a coin holder can challenge this transaction after the fact, and the adversary will not be able to produce a valid signature authorizing the transaction, and so the coin owner can be appropriately punished for allowing the illegal transaction.

Since WhoPay also aims to protect anonymity, and the base PPay system allows some attacks by users which are detectable, but not preventable in realtime, it may be of concern that criminal use could go unpunished. However, since the WhoPay system also has the fairness property, any crime that is detected can be appropriately punished after revoking anonymity.

Anonymity. WhoPay prevents a coalition of dishonest entities from proving the identity of a coin holder at any point after it has been issued (thus a user who wishes to remain anonymous would be wise to obtain coins via issue, or issue coins they purchase to themselves), unless those entities include a corrupt judge.

Upon initial purchase, the coin holder and owner are identical. Since coin holders are known to the broker, there is no anonymity. Let us assume the coin is then issued to an honest user u , and there exists another honest user v . Were an adversary to include collusion among all users besides u and v , and the broker, it still would only have access to the public key chosen by u for the coin, and the private group key signature of u . The public key is randomly chosen, and hence is indistinguishable from a random choice by v . From the properties of group signatures, a group signature by u is computationally indistinguishable from a group signature by v , since the adversary does not have access to the judge's private key. Similar information is made available to outside parties during issue, deposit, and renewal, and thus by similar construction no possible transactions after the initial purchase will allow any but the coin holder to know his own identity with certainty.

Note that so far we have been talking about anonymity in terms of application level identities such as those encoded in public key certificates. In many situations network level identities (e.g., IP addresses) can convey a lot of information and are hence worth hiding as well. There have been many studies in this area, most of which, such as Onion Routing [18] and Tarzan [9], involve hiding end points IP addresses by using third party proxies. In this paper, we will assume such mechanisms will be adopted whenever network level anonymity is desired.

Fairness. Recall that fairness means the broker and the judge, working together, can reveal the identities of all parties involved in a particular transaction without learning any information about other transactions. Transactions signed with (non-group) private keys expose signer identities and are automatically fair. For those signed with group private keys, the broker sends the transactions of interest to the

judge, who recovers the identities of the signers of these transactions and sends them back. Note that no information about other transactions is learned in this process. Thus WhoPay is fair.

Transferability. Recall that transferability means the recipient of a coin can use the same coin to pay another user without identifying himself to the broker. In WhoPay, when the coin owner is online, broker is not involved in coin transfers and hence does not learn the identity of the payer. In fact, even the coin owner does not learn the identity of the payer, due to the anonymity property mentioned above. When transferring an offline coin via the broker, the payer also remains anonymous throughout the transaction. Thus WhoPay supports transferability.

Scalability. During the lifetime of a WhoPay coin, there is one purchase, one issue and one deposit, but there could be an arbitrary number of transfers and renewals. Thus we expect transfers and renewals to dominate the system load. Transfer and renewal load is distributed across peers. In general, the more coins a peer issues, the more transfers and renewals he needs to handle. This is desirable, as we expect more active peers to do more work. The broker is only involved in coin purchases, deposits, synchronizations and downtime transfers/renewals. The load generated by the last three items depends on the availability of peers, but we expect the majority of transaction load is handled by the peers rather than by the broker and the broker load increases sublinearly as the number of peers (or the total system load) increases. We will run simulations to study scalability in detail in Section 6.

5 Extensions

5.1 Real-time Double Spending Detection

By making sure all fraud will eventually be detected and punished, WhoPay as described so far already provides a level of security as good as PPay. One might be concerned that detecting fraud until coin deposit time may be too late and much damage could have been done by that time. To address this issue, WhoPay also provides **real-time double spending detection** by providing a publically viewable coin binding list. This provides a check for coin owners. A peer can verify that the relevant public binding has been properly updated before accepting payment. Each peer can also monitor public bindings for its own coins to detect malicious updates.

The major challenge is how to implement this public coin binding list. Publishing and serving all the bindings in a central trusted server would defeat the purpose of a distributed system. Also, giving any one peer too much control over where the bindings are published compromises security. We propose to publish the coin bindings in a trusted,

access-controlled distributed hash table (DHT) infrastructure. Like hash tables, distributed hash tables provide a put/get interface for storing/retrieving values under given keys and are distributed across a network. To do so, DHTs use a routing algorithm that ensures that a query under a given key is always routed toward the same host in the network. CAN [16], Chord [24], Pastry [21] and Tapestry [11] are early examples of DHT.

Naturally, only the owner of a coin should be allowed to write to the coin's binding, while anyone can read the binding. Recall that the coin bindings are keyed by public keys, such as pk_{CU} . The DHT should be designed in such a way that only users who know sk_{CU} can write to the id pk_{CU} (by providing the right signature, which can be published along with the binding to back it up), but anyone can read the id pk_{CU} . This way, any user can verify the binding but only the owner of the coin can modify this binding. To allow the broker to take over during downtime, the broker should also be allowed to write to any id. By allowing the broker to update the bindings in the public list, real-time double spending detection will continue working during the owner's downtime. To monitor this DHT-based public binding list, peers can either poll the bindings of interest periodically or use a register/notify mechanism such as Bayeux [27], Scribe [4, 5], or CAN-mc [17].

We understand that there is a huge amount of trust placed in this DHT infrastructure for its access control and register/notify service. To address this issue, we can either assume this infrastructure is provided as a service by a trusted entity (e.g., AT&T), or in the case that this infrastructure consists of arbitrary members and lacks administrative control, introduce mechanisms to detect and remove misbehaving nodes. Either way, further study is needed.

5.2 Issuer Anonymity

As pointed out in Section 4, the identity of the payer is exposed during coin issues. We identify three approaches to this issue. First is for users to adapt their spending habits around this. Peers can issue coins to pay for less sensitive items or services. When anonymity matters, peers should choose to transfer instead of issue coins. As long as peers have enough coins to transfer, this should not be a major concern.

Secondly, we can introduce *coin shops* into the system. Coin shops purchase coins from the broker, and peers purchase coins, using the issue procedure, from the coin shops. The only transactions a coin shop performs is to purchase coins from the broker, to issue coins to peers, and to manage (i.e., handle the transfers and renewals of) the coins it has issued. Coin shops do not care about anonymity; they are in this business for profit, e.g., by charging a small fee for each coin issued. Peers do not own, and hence never

issue coins. Peers spend coins only using the transfer procedure, which is anonymous.

The third approach is more complicated. The lack of issuer anonymity stems from the encoding of coin owner identity in coins. So, we propose to remove this information so that a coin owner cannot be identified given a coin. Because we represent coins as public keys, a peer can show its ownership of a coin by proving its knowledge of the private key, so it is not necessary to explicitly state the owner. Thus, a coin now has the initial form of

$$C = \{pk_{CU}\}_{sk_B}$$

instead of $C = \{U, pk_{CU}\}_{sk_B}$, for example. Once issued, the coin becomes:

$$Coin = \{C, pk_{CH}, seq, exp_date\}_{sk_U}$$

where only C has a different format now, but everything else stays the same as before.

The explicit coin owner information encoded in coins was originally used in three places. First, when peers transfer coins, the payer needs to contact the coin owner to request the transfer. Second, when peers perform synchronization with the broker, the broker needs to map coins to owners in order to determine which coins' state to send to peers. Third, when certain fraud (e.g., double issuing) is detected, coin owners should be held responsible. By removing coin owner information from coins, we break these three things. Now we will present solutions to these problems such that WhoPay can still operate properly.

Our solution to the first problem is to use an anonymous indirection mechanism like the Internet Indirection Infrastructure, or *i3* [23]. *i3* is an overlay network consisting of *i3* servers that store triggers and forward messages. Each coin now includes a handle and peers send messages to this handle when they want to contact the coin's owner. That is, coins now have the initial form of $C = \{h_{CU}, pk_{CU}\}_{sk_B}$, where h_{CU} is the handle of the coin. The coin owner registers a trigger on this handle so that all messages sent to this handle will be forwarded. These handles act as pseudonyms to obscure the identity of the coin owner. Note with the use of this indirection mechanism, the issue protocol and the transfer protocol look exactly the same from the payee's point of view, and thus the payee cannot tell whether or not the payer is the coin owner.

A simple, but inefficient solution to the second problem would be for the peer tell the broker which coins it owns, which could be a long list. Moreover, to ensure the secure and correct functioning of the system, the broker must engage in a large number of proofs of coin ownership, which are expensive both in communication and computing. A better alternative, inspired by the observation that synchronization is needed if and only if the public binding for a

coin and the local binding of the coin owner are different, is to use **lazy synchronization**. Instead of synchronizing everything immediately after rejoining the system (which we call *proactive synchronization*), the peer waits until it is absolutely necessary, i.e., when a transfer or renewal request is received. Upon receiving such a request, the coin owner checks the relevant binding in the public coin binding list and updates its local binding if it is outdated (we will refer to this operation as simply a *check*). This way, the involvement of the broker during synchronization becomes optional

The last problem is one of fairness. Thus, we simply extend the use of group signatures which provide fairness elsewhere in the system to the issue messages as well. Group signatures allow the issuers to remain anonymous under normal circumstances, while making sure that they will get caught and punished if they cheat.

6 Simulation

6.1 Simulation Setup

We have run simulations to measure load distribution, especially as load grows. In order to evaluate different models of use, we used three user behavior policies in our simulation. Each policy describes a different order of preferences for payment methods, reflecting differing biases towards either broker load or user protection. We cover the two policies which reflected the extremes and had more interesting results:

Policy I	Policy III
Transfer online coin	Transfer online coin
Transfer offline coin	Issue existing coin
Issue existing coin	Purchase and issue
Purchase and issue	Deposit, purchase, issue

In policy I, each peer tries to get rid of coins received from other peers as quickly as possible. The motivation behind this might be fear of fraud. Policy III simulates the best case in terms of broker load: each peer tries to avoid dealing with the broker as much as possible. These two policies are also different in the way they deal with offline coins. Policy I chooses to transfer offline coins through the broker, and the motivation for doing so might be that each peer wants to minimize the number of coins it needs to manage. In policy III, peers deposit offline coins, and purchase new coins to issue. We suspect that doing this effectively moves the ownership of the coins from an offline peer to an online peer, and may reduce the load on the broker in the long run. For these reasons, we call policy I the user-centric policy, policy III the broker-centric policy.

Our simulations study load distribution with different peer availability, different spending policies, lazy synchronization vs. proactive synchronization, and different numbers of peers. To study the first three, we use the following setup, which we refer to as Setup A. There are a total of 1000 peers. Peers join and leave the system: session lengths follow exponential distribution with mean μ for on-line transactions and ν for offline. Availability of peers can be roughly indicated by the value $\alpha = \mu/(\mu + \nu)$. To model different peer availability, we run three sets of simulations, with ν set to 1, 2, or 4 hours. We call these three sets of simulations *short downtime simulation*, *median downtime simulation*, and *long downtime simulation*, respectively. In each of these simulations, we further vary μ from 15 minutes to 32 hours. For each peer, candidate payment events arrive as an independent Poisson process with rate 1 payment per 5 minutes, with the payee selected randomly. A candidate payment event will result in an actual payment event if and only if the randomly selected payee is online at the time, therefore the actual payment events (for each peer) form an independent Poisson process with rate α payments per 5 minutes. We use a renewal period of 3 days, and each run lasts for 10 days.

To gain insights into how the system scales with increasing number of peers, we run another set of simulations, which we refer to as Setup B. In these simulations, we vary the size of the system from 100 peers up to 1000 peers. We fix the mean online and offline session lengths to 2hrs, i.e. $\mu = \nu = 2$ hrs, simulating a 50% peer availability. The rest of the configuration stays the same. The setups are summarized in table 1.

Following are the simulation results. Due to space limitation, we show only the results for the median downtime simulation (results from short and long downtime simulations are similar) with lazy synchronization.

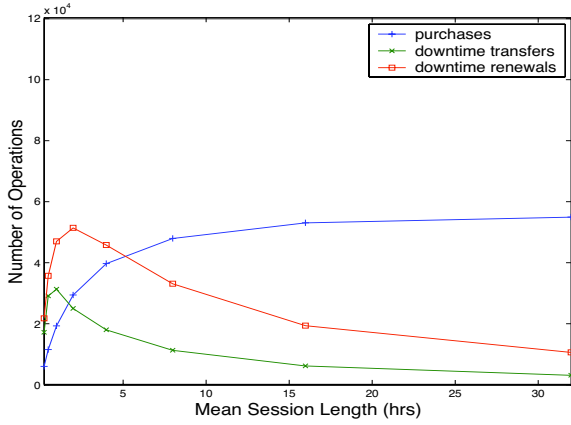
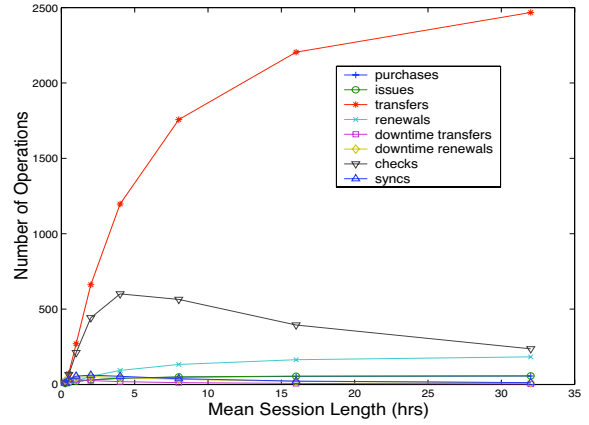
6.2 Simulation Results

Load distribution. The WhoPay system is built from the following coarse-grained operations: coin purchases, issues, transfers, deposits, renewals, downtime transfers, downtime renewals, synchronizations, checks, and lazy synchronizations. Here we only analyze load distribution in terms of these operations; another paper gives our estimates of the (CPU and communication) costs of these operations and analyzes the aggregate load distribution.

Under policy I with proactive synchronization, the broker needs to handle purchases, downtime transfers, downtime renewals, and synchronizations. Figure 2 shows the broker load in terms of these operations with lazy synchronization. As peer availability increases, greater activity leads to increased payment events. However, fewer transactions involve offline peers and need to go through the broker.

Table 1. Simulation Setup

Setup	Policy	Sync	μ	ν	Number of peers
A	I, II.a, II.b, III	proactive, lazy	15 mins - 32 hrs	1 hr, 2 hrs, 4 hrs	1000
B	I, II.a, II.b, III	proactive, lazy	2 hrs	2 hrs	100 - 1000

**Figure 2.** Broker Load: Policy I + Lazy Sync**Figure 3.** Average Peer Load: Policy I + Lazy Sync

The trend of broker load thus reflects the combined effect of these two competing forces. For purchases, the first force dominates since the number of purchases increases as peer availability increases. For downtime transfers and downtime renewals, the first force dominates when peer availability is low while the second dominates when it is high, resulting in a growing, then decaying curve.

Unsurprisingly, peer load shows an inverse relationship to broker load: average peer load rises as peer availability increases (see Figures 3), for the same reason that broker load drops. One striking point though, is that under all configurations, transfers dominate peer load. Both the broker-centric policy and lazy synchronization cut down broker load significantly. In summary, as peer availability increases, broker load decreases and peer load increases; transfer dominates peer load and transfer-via-owner is the dominant payment type.

7 Related Work

We got the idea of using public keys to represent coins from the Burk-Pfitzmann anonymous transfer system [2]. The Vo-Hohenberger scheme [25] adds fairness to Burk-Pfitzmann with the use of group signatures. Both are online transfer systems, as is WhoPay; but while WhoPay distributes transfer load across peers, each transfer in Burk-Pfitzmann and Vo-Hohenberger needs to go through a central entity.

An alternative to these online transfer systems, quite naturally, is offline transfer systems. For example, peers can

transfer coins by using layers: each time a coin is transferred, the current holder of the coin simply adds another layer of signature to the coin, which serves as a proof of relinquishment. Group signatures can be used to provide fairness without compromising anonymity. No third party is involved in the transfer and thus the scheme is extremely scalable. This scheme suffers two major problems though. First, coins grow in size after each transfer. Second, double spending is easier to commit and harder to defend than in online transfer systems. It has no real-time double spending detection. Anyone can double spend in this scheme, while in WhoPay only coin owners can double spend. Nonetheless, layered coins can be a lightweight alternative to transfer-via-broker when coin owners are offline. To alleviate the size and security problems mentioned above, a maximum number of layers can be imposed.

Micropayment schemes are designed to handle payments of small amount, e.g., less than \$5. These schemes must be lightweight, otherwise the cost will outweigh the value of the payment. Their basic approach is to aggregate many small micropayments into a few bigger payments. Early examples include PayWord [20] and Electronic Lottery Tickets [19], both of which use secure hash chains, albeit in different ways. These algorithms, however, only allow aggregation by an individual merchant and thus are limited by the frequency of a given consumer's purchases with that merchant. More recently, schemes have been designed to allow aggregation across multiple consumers and multiple merchants. These schemes generally involve a third party payment service provider that sits between consumers and

merchants and performs the aggregation for the merchants. Some of these schemes, including BitPass [1], Firstgate [8] and Paystone [14], require pre-enrollment or pre-deposit of funds with the payment service provider, while others, including PepperCoin [15], don't.

While WhoPay is not specifically designed as a micropayment scheme, it can certainly be extended to support micropayment. For example, we can use a scheme such as PayWord to first aggregate small micropayments into bigger payments and carry out the bigger payments using WhoPay. That is, each pair of users maintains a soft credit window between themselves and only makes payments when this window reaches a threshold value.

8 Conclusions

An electronic payment system ideally should provide security, anonymity, fairness, transferability and scalability. Existing payment schemes often lack either anonymity or scalability. In this paper we proposed WhoPay, a peer-to-peer payment system that provides all the above properties. For anonymity, we represent coins with public keys; for scalability, we distribute coin transfer load across all peers, rather than rely on a central entity such as the broker. This basic version of WhoPay is as secure and scalable as existing peer-to-peer payment schemes such as PPay, while providing a much higher level of user anonymity. We also introduced the idea of real-time double spending detection by making use of distributed hash tables (DHT), which further improves the security level of WhoPay. Through simulations, we have shown that WhoPay should scale well under typical operating conditions.

A trusted DHT infrastructure that supports access control and a register/notification mechanism is essential to WhoPay's real-time double spending detection mechanism. More work in this area is needed.

References

- [1] Bitpass. <http://www.bitpass.com>.
- [2] H. Burk and A. Pfizmann. Digital payment systems enabling security and unobservability. *Computers and Security*, 8:399–416, 1989.
- [3] J. Camenisch, J. Piveteau, and M. Stadler. An efficient fair payment system. In *Third ACM Conference on Computer and Communication Security*, pages 88–94, 1996.
- [4] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications (JSAC) (Special issue on Network Support for Multicast Communications)*, 2002.
- [5] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scalable application-level anycast for highly dynamic groups. In *Proceedings of NGC 2003*, September 2003.
- [6] D. Chaum. Blind signature system. *Advances in Cryptology*, 1983.
- [7] D. Chaum and E. V. Heyst. Group signatures. *Lecture Notes in Computer Science*, 547:257–265, 1991.
- [8] Firstgate. <http://www.firstgate.com>.
- [9] M. Freedman, E. Sit, J. Cates, and R. Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings for the 1st International Workshop on Peer-to-Peer Systems*, 2002.
- [10] D. Geer. E-micropayments sweat the small stuff. *Computer*, 37(8), August 2004.
- [11] K. Hildrum, J. Kubiawicz, S. Rao, and B. Zhao. Distributed object location in a dynamic network. In *Proceedings of the Fourteenth ACM Symposium on Parallel Algorithms and Architectures (SPAA'02)*, August 2002.
- [12] Kazaa. <http://www.kazaa.com>.
- [13] Paypal. <http://www.paypal.com>.
- [14] Paystone. <http://www.paystone.com>.
- [15] Peppercoin. <http://www.peppercoin.com>.
- [16] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM*, September 2001.
- [17] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. In *Proceedings of NGC 2001*, 2001.
- [18] M. Reed, P. Syverson, and D. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communication Special Issue on Copyright and Privacy Protection*, 1998.
- [19] R. Rivest. Electronic lottery tickets as micropayments. In *Proceedings of Financial Cryptography Conference*, 1997.
- [20] R. Rivest and A. Shamir. Payword and micromint: Two simple micropayment schemes. In *Proceedings of 1996 International Workshop on Security Protocols*, 1996.
- [21] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, November 2001.
- [22] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.
- [23] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *Proceedings of ACM SIGCOMM*, August 2002.
- [24] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM*, September 2001.
- [25] B. Vo and S. Hohenberger. A fair payment system with online anonymous transfer. Masters thesis, to appear, December 2006.
- [26] B. Yang and H. Garcia-Molina. Ppay: Micropayments for peer-to-peer systems. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS)*, 2003.
- [27] S. Zhuang, B. Zhao, A. Joseph, R. Katz, and J. Kubiawicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *The 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, 2001.