Anonymous Atomic Transactions

Jean Camp, Michael Harkavy,
J. D. Tygar, Bennet Yee

July 1996
CMU-CS-96-156

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We show here an example of a protocol that satisfies anonymity properties while providing strong ACID (atomic, consistent, isolated, durable) transactional properties, resolving an open question. Blinded signatures are used to certify an anonymous asymmetric key which authorizes the use of a specified value. A public transaction log is used to support a linear commitment scheme for transactions in which the value authorized by the blind signed key is spent.

Sponsored by the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and ARPA under grant number F33615-93-1330, National Science Foundation cooperative agreement IRI-9411299, the US Postal Service, and Visa International. Bennet Yee is in the Computer Science Department at the University of California, San Diego. Jean Camp is in the Engineering and Public Policy Department at Carnegie Mellon University.

Keywords: cryptography, electronic currency, digital signatures, anonymous currency, atomic transactions

1 Introduction

Consumer privacy is an important goal of electronic payment systems. Some researchers have approached this question by adopting a token-based model. These tokens are meant to act as a type of currency: they can be used to purchase a good, but like coins, they do not reveal the identity of the holder. These systems offer privacy in making a purchase. Some typical examples of token-based electronic payment protocols ("digital cash" protocols) are [2, 3, 7, 5, 14]. These protocols provide consumers with the ability to make anonymous purchases, purchases which can not be tracked by a bank to identify the purchaser. A stronger form of anonymity can be considered — anonymity in which the identity of the purchaser is hidden from both the bank and the merchant selling the goods.

But what happens when things go wrong? If the network (or merchant server) goes down during a purchase, how can users complain about non-delivered goods? If their purchases are anonymous, how can they prove that they really did pay and did not receive the goods? How can electronic judges and merchants adjudicate these complaints? How can they determine whether the consumer was really denied the goods, or whether the consumer is just trying to illegitimately acquire merchandise for free? And how can a consumer obtain satisfaction when the purchase is anonymous? These questions are especially important because the Internet today is an unreliable network — anyone who has spent some time browsing the web knows that communications often fail. Unscrupulous consumers and merchants will certainly attempt to take every advantage of system failures.

To illustrate the problem, consider the following simplified digital cash protocol: consumers pay for electronic goods with tokens. These tokens are anonymous, but designed so that if the consumer ever uses the same token twice, the consumer's identity is revealed. Suppose a consumer pays for a good, but before she can receive acknowledgment that the merchant received payment, the network fails. Now, what can the consumer do? She doesn't know whether the merchant received the payment or not. She has two basic strategies:

- She can *spend* the token again, by returning her token to the bank or spending it with a second merchant. But then, if the first merchant really did receive the token, she may be creating a race condition. Whoever gets the token to the bank first will get the money. Worse, when both tokens do reach the bank, the consumer will be accused of double-spending. Now, one can imagine variations on the digital cash protocol where a consumer might file a special type of complaint with a bank, but the design of this variation is non-trivial. Most types of variations will either reveal the consumer's identity, allow a new type of fraud, be subject to ambiguous results if a message is not delivered, or have other undesirable effects. This topic was addressed at length in [4, 16, 17].
- She can *wait* and not spend the money. But in this case, the consumer has locked up her funds. If the merchant did not receive her payment, then the consumer may be waiting for a very long time!

A standard approach to addressing the question of reliability is the notion of ACID (atomic, consistent, isolated, durable) transactions [10]. In the distributed systems community, ACID transactions have been widely adopted as the standard mechanism for realizing distributed transactions. ACID transactions are the payment transactions should be *failure-atomic*, so that failures in parts of the system will not leave the entire system in some ambiguous, intermediate state.

How can we interpret these transactions in the context of electronic commerce? Tygar [16] has proposed using the classification below. Tygar began by assuming a model where consumers are purchasing electronic goods and services that will be delivered over a network (such as WWW page, for example). For tangible physical goods, alternative definitions are required to properly satisfy the atomicity property (motivating a multi-billion dollar industry in tracked, receipted courier delivery of messages and packages!) Tygar defines three classes of atomicity for electronic goods.

- **Money atomic** transactions feature atomic transfer of electronic money — the transfer either completes entirely or not at all. In money atomic protocols, money is not created or destroyed by purchase transactions.

- **Goods atomic** transactions are money atomic and also ensure that the consumer will receive goods if and only if the merchant is paid. Goods atomic transactions provide an atomic swap of the electronic goods and funds — similar to the effect of "cash on delivery" parcels.

- **Certified delivery** protocols are goods atomic and also allow both the consumer and merchant to prove exactly what was delivered. If there is a dispute, this evidence can be shown to a judge to prove exactly what goods were delivered.

Using this classification, we can see that the simplified digital cash protocol described above is *not* money atomic. The obvious question is: are anonymous atomic transactions possible? [16] This has been an open question.

Our first attempt to solve this question would be to use standard techniques to make a digital cash transaction atomic. The standard method for doing this is *two-phase commitment* [1, 9, 10, 11, 12]. In short, in two-phase commitment, one party assumes the role of transaction coordinator. That party knows and records the identities of all other parties in a non-volatile log. Each of the parties records its state before the transaction begins. As the transaction moves forward, various parties complete their required computation. Before changing the permanent store of those values, the parties send a message to the coordinator indicating that they are ready to commit. (Alternatively, they may abort the transaction by sending a negative message to the coordinator.) After receiving ready messages from all parties, the coordinator issues a commit message to all parties, causing the transaction to become permanent. Alternatively, if the coordinator receives an abort request or if the coordinator can not establish contact with one of the parties, the coordinator can abort the transaction by sending an abort message; in that case, all parties reverse the computation that they conducted towards the transaction.

So, as we see, the two-phase commit protocol requires that at least one party participating in the protocol (the transaction coordinator) knows the identity of all the parties involved. Additionally, two-phase commit assumes a *fail-stop* fault model, where the parties to the protocol can fail by stopping due to a crash, but not by lying or otherwise trying to cheat. In electronic commerce protocols, of course, we must be able to tolerate arbitrary Byzantine faults; one way to do this is to provide sufficient auditing information to detect these faults and later assign responsibility. This makes the standard two-phase commit protocol inappropriate for use in anonymous electronic commerce systems.

1.1 Our contribution

Let us return to the open question mentioned above: Can anonymous transactions be atomic? Many researchers have speculated that the answer is negative. However, in this paper, we reverse this commonly held belief by answering the question affirmatively. We present a set of protocols for electronic commerce transactions which combines anonymity and atomicity while requiring very limited trust assumptions. We prove the goods atomicity properties of our protocols. For certified delivery we provide two variations:

- **one-sided certified delivery** where the consumer can prove what goods were delivered in case adjudication is needed (e.g., the goods do not match their description). The merchant is also guaranteed to be paid if and only if the consumer successfully obtains the electronic goods. On the other hand, the merchant can not prove that the consumer successfully received the goods promised. (This is the protocol presented in Section 3.3. As we argue below, if the burden of proof is on the consumer, then this method suffices to allow the consumer to prove the results of the transaction.)
- **two-sided certified delivery** which provides proof of the delivery of specific contents to both parties.

As we discuss below, there are tradeoffs between these two properties — two-sided certified delivery still keeps the identity of the consumer anonymous, but it does reveal some information about a cryptographic key used by the consumer. (This normally is not a problem since a consumer is expected to choose unique cryptographic keys for each transaction. However, we do discuss reusable keys below in Section 8.1, and if this option is chosen, there is a shade of privacy lost. This variation is presented in Section 8.6.)

It is important to emphasize that we have not designed this protocol with performance considerations in mind. The efficiency of digital cash protocols is already controversial [15, 16] and our mechanisms also require large amounts of computation. The intellectual contribution of this paper lies in showing that the widely held belief that atomic transactions can never be anonymous is not correct.

Section 2 explains our basic cryptographic and system assumptions. Section 3 presents a high level overview of the design of our protocols, followed by a more detailed description of our protocols. Section 4 informally analyzes the correctness of the protocols, and gives an argument for verification of the one-sided certified delivery property. Section 5 describes the logging requirements for all the parties in our system. Section 6 details the privacy properties of our system, giving an exhaustive list of the types of information available to the various parties. In Section 7, we review the basic trust assumptions required by our system and show that strengthening these assumptions leads to simplified versions of the system. Section 8 presents several variations to our basic protocols, allowing partial spending of the withdrawn amount, greater efficiency by permitting public key reuse, etc. Finally, Section 9 concludes the paper.

2 Assumptions

In designing our protocols, we made a variety of assumptions about the capabilities of the participants and made use of several cryptographic tools. This section describes our assumptions and introduces some of the important tools which will be used.

Four parties participate in our protocol: a consumer C, a merchant M, a bank B, and a transaction log L. Now, of course, our parties are designed to allow multiple, scalable, simultaneous transactions that do not interfere with each other — this is the *isolation* requirement in ACID transactions. Thus, there can be many more than four parties — but in a single transaction, there will only be four parties.

All parties can perform basic cryptographic operations, e.g. cryptographic hash computation, signature computation and verification. All parties have well-known or verifiable public keys. All signatures can be verified by the receiving party.

To justify the claim of anonymity, the identity of the consumer must be protected from all other parties (that is, we make assumptions of *strong anonymity*).

We assume that the communications channels (in particular those between the consumer and the other parties) are anonymous, i.e. no information about the identity of the consumer is gained by communicating with the consumer. Now, in practice, this may be a questionable assumption — can't a merchant or a third party infer the consumer's identity through the TCP/IP return address on his packets? Supporting this assumption in an implementation may involve a fifth party, an anonymizer, which the consumer trusts not to reveal identity information.[8]

To provide atomicity, a modified, cryptographic version of the two-phase commit protocol is implemented using an external, publicly accessible transaction log. The transaction log receives and records messages, and then reproduces the recorded messages. The log localizes the global commit decision to a single entity. The log also acts as a time-keeper, determining when to abort transactions due to a time-out.

Communication channels are assumed to be secure. The method used to implement this security (e.g. public keys) is not important to the functioning of the protocol. Some messages could be left unsecured, improving efficiency at the cost of a limited loss of privacy, but the details of this are not currently addressed.

Chaum [6] made a critical discovery that enabled the idea of digital cash to take place: *blinded tokens*. The use of blinded tokens is essential to our protocols. A token is a piece of data which can be created only by a specific issuer. Creation of a token without assistance from the issuer should be computationally infeasible. Blinded tokens are created by an interaction of a consumer with the issuer (the bank). After the interaction, the consumer has knowledge of a token which the issuer can not specifically identify. That is, the issuer does not know which token (from the range of valid tokens) the consumer has obtained. The tokens used in this protocol will have the additional property that each token, denoted Q^*, specifies the public half (including modulus), denoted Q, of a public key pair.

It is assumed that the consumer has an account with the bank, and that the bank can mint blinded token currency. This requires the bank to maintain token information as detailed in Section 5.

The transaction protocol given assumes minimal preparation and delivery costs for the goods. Goods must be prepared and delivered, although not in a usable form, prior to any guarantee of payment to the merchant.

Message signatures are computed on hash values of the plaintext, and then appended to the plaintext to form a signed message. This is relevant in the first step of the purchase protocol, P1, for efficiency reasons. It is also relevant in the second step, P2, so that the bank can read Q^* in order to determine Q. This assumption can be dropped with minor changes to these steps.

3 Protocols

In this section, we first give a high level overview of our anonymous atomic transaction system design, and then describe our abstract protocols.

Critical to our system is the use of a blind signature in the withdrawal protocol. Here, the consumer obtains a blinded token from the bank as a result of withdrawing money from the consumer's account.

Unlike previous works where the blind-signed data is a token which represents value, in our protocol the public key of a newly generated public/private key pair is signed; this certifies a trapdoor function rather than data to be disclosed in the purchase protocol. This effectively provides a temporary, anonymous certificate of ownership of the withdrawn amount. The private key of the key pair is known only to the consumer, and it is used with the certificate to anonymously authorize transfers of the withdrawn amount to a merchant's account. Authorization messages signed with this key are used in our transactions to signal readiness to commit to a purchase transaction, and to serve as part of the "paper trail" to prove that the token has been expended.

In the purchase phase, the merchant delivers encrypted goods along with a signed contract providing the goods description and the price. If the consumer finds the contract acceptable, readiness to commit is sent to the bank in the form of a signed message (using the above blind-certified key) to authorize the transfer of funds if the transaction commits, and then the bank similarly signals its readiness to the merchant with a message promising an anonymous deposit into the merchant account when the transaction commits. The transaction commits when the transaction log records a message from the merchant which contains the merchandise key.

Timely delivery of the merchant's message to the transaction log results in the transaction committing, thereby crediting the merchant's account and releasing the merchandise key to the consumer. If the merchant's message does not arrive before the expiration time, the transaction aborts.

Note that unlike standard two-phase commit, there is no central transaction coordinator; instead, the various parties' readiness to commit are determined using non-repudiable messages in a distributed, cascading fashion as explained in section 4.

Next, we give a detailed description of the withdrawal and purchase protocols.

3.1 Notation

We use the following notation to describe steps in a protocol.

1. $X \rightarrow Y$ *messagetext* — *label*

Here, the step number of the message is given (this is the first message in the protocol), the message is sent from X to Y, the text of the message is *messagetext*, and the step is named *label*.

We use the notation $(\textit{message})_p$ to indicate the *message* is signed with public key p, and $E_k(\textit{field})$ to indicate that a field is encrypted with symmetric key k.

3.2 Withdrawal and Exchange

The consumer generates a public key pair to use with each withdrawal. The public half of the pair is used to form *blinded-request*.

W1. $C \rightarrow B$ $(\textit{blinded-request})_c$ — $\textit{withdraw}_c$

W2. $B \rightarrow C$ *signed-blinded-request* — $\textit{withdraw}_b$

After W2, the consumer can form the token Q^* by unblinding *signed-blinded-request*. The contents of Q^* specify the public key Q (including modulus), whose private half is known only to C.

A token may be anonymously exchanged for a new token in a similar fashion by replacing the consumer's public key with the token's single-use key.

E1. $C \rightarrow B$ $Q^*, (\textit{blinded-request})_q$ — $\textit{withdraw}_c$

E2. $B \rightarrow C$ *signed-blinded-request* — $\textit{withdraw}_b$

The following is an example protocol using a specific blinding technique. The bank has an RSA public key pair with modulus N_t, public exponent 3, and private exponent $t = 3^{-1} \bmod \phi(N_t)$. The bank has also made public a cryptographic hash function h.

1. C generates a desired public key pair Q, q with modulus N_q
2. C selects a random number $r \bmod N_t$
3. W1. $C \rightarrow B \quad h(Q) \cdot r^3 \bmod N_t$
4. B computes $(h(Q) \cdot r^3)^t \equiv h(Q)^t \cdot r \bmod N_t$
5. W2. $B \rightarrow C \quad h(Q)^t \cdot r \bmod N_t$
6. C computes $r^{-1} \bmod N_t$ and then $h(Q)^t \bmod N_t$
7. C has $Q^* = Q, (h(Q)^t \bmod N_t)$

The bank may use multiple signature keys for its blind signature, corresponding to different *brands* of tokens. The brand of a token determines its denomination and its withdrawal date. Having token brands is important for limiting the data logging requirements for the bank: until a brand of blinded token is withdrawn from use, the bank must maintain a database containing auditing information proving that expended tokens have been spent in order to prevent double spending (see Section 5). By *a priori* declaring that tokens will be worthless after the brand withdrawal date, the bank limits its data logging obligations; furthermore, brand withdrawal will also limit risk, since it limits the amount of time attackers will have to attack the key. Next, we discuss how the blind-signed token obtained above is used in the purchase protocol.

3.3 Purchase

Some negotiation of the transaction contract is assumed to take place prior to the transaction steps listed below. Given the current approach, the method of this negotiation has no direct bearing on the protocol. As above, each message is annotated with a mnemonic which describes the purpose of the message, and which will be used to refer to the message. For example, *authorization* denotes the authorization action by the party identified in the subscript. The protocol is an example of linear commitment is the sense defined in [10].

P1. $M \rightarrow C \quad (n, \mathit{contract}, E_k(\mathit{goods}))_m$ — goods_m

P2. $C \rightarrow B \quad (n, \mathit{expiration}, M, L, Q^*)_q$ — $\mathit{authorization}_q$

P3. $B \rightarrow M \quad (n, \mathit{expiration}, M, L, \mathit{value})_b$ — $\mathit{authorization}_b$

P4. $M \rightarrow L \quad (n, \mathit{expiration}, k)_m$ — $\mathit{authorization}_m$ or

P5. (a) $L \quad ((n, \mathit{expiration}, k)_m)_l$ — commit

(b) $L \quad ((n, \mathit{expiration}, M, \text{failed})_l$ — abort

In step P1, the merchant sends a signed copy of the contract and goods to the consumer. It is essential that the contract (*contract*) contain a description of the goods in order to provide one-sided certified delivery. The goods (*goods*) are encrypted with a single-use private key (k), referred to as the merchandise key. The merchandise key will be revealed to the log and the consumer if the transaction commits. The message includes a transaction number (n) generated by the merchant which should be different from any previously generated transaction number from the same merchant. A duplicate number could be detrimental only to the merchant. Upon receipt, the consumer verifies that the contract is acceptable.

In step P2, the consumer decides upon an expiration time (*expiration*) for the transaction, after which the transaction is considered to have failed, and the token can be reused or replaced. The consumer also selects a transaction log (L) for the transaction. Both the bank and the merchant have effective veto power over the consumer's selection of L and *expiration*, since they will not provide authorization before knowing these values. The consumer then signals to the bank its readiness to commit by specifying the transaction and the token and key to be used (Q^*). The bank must verify the validity of Q^*, including a check against reuse. The bank then uses Q^* to check the message's signature.

In step P3, the bank tells the merchant that it and the consumer are ready to commit, and includes in the message the value of the token (*value*) to be used for payment. The merchant verifies the transaction number is correct and that the token value, log identity, and expiration time are acceptable.

In step P4, the merchant commits to the transaction by sending the merchandise keyto the log, along with the time-out, transaction number, and a signature. Upon receipt, the log verifies that the expiration time (*expiration*) has not passed.

In step P5a, the log records the merchant's commitment if and only if it was received before *expiration*. The precise method of distribution of the recorded message is not important to determining atomicity properties, but some method for providing timely, good-faith delivery to the consumer is of practical importance. Any party can use this log record in conjunction with other signed messages obtained during the transaction to force the bank to transfer funds or to obtain the goods decryption key, completing the transaction.

In step P5b, which may occur only after the *expiration*, the log generates a negative authorization at the request of the consumer or the bank. This indicates that no P4 for the given merchant and transaction number was received prior to the given expiration time. This allows the token to be freed for reuse or exchanged following a failed transaction; after *abort* is generated, the purchase protocol terminates.

4 Correctness and Atomicity

The atomicity properties of the protocol rest on the atomicity of the transaction log's non-repudiable *commit* (or *abort*). The transaction log will eventually produce exactly one of *commit* or *abort* for any transaction. The other parties can use this, along with other data gathered during the course of the transaction, to prove that the transaction did (or did not) complete. Conversely, if a party claims that the transaction did (or did not) complete, it must be able to provide this proof to justify its claim to other parties.

The transaction protocol follows the two-phase commit model, however, the authorization actions of the parties are cascaded. First, the consumer authorizes the bank to lock the token to the transaction. Next, the bank authorizes the merchant to transmit the key to the log. Then, the merchant sends the merchandise key to the log, authorizing the log to commit the transaction. Finally, the log issues the global commit. Accountability is similarly cascaded so that a party can be held accountable exactly when it has made an authorization and the transaction commits. For example, if the bank authorized the merchant to deposit the key ($authorization_b$) without having received the consumer's authorization to lock the token ($authorization_q$), then the bank could be held accountable by the merchant (who would have *commit* and $authorization_b$), but the consumer could not be held accountable by the bank (which would have *commit* but not $authorization_q$).

The merchant can use *commit* and $authorization_b$ to prove that the bank should credit the merchant's account with the value of the token. This provides the bank with *commit*. Possession of *commit* assures the bank that it will not be subject to a claim that the transaction has failed, since such a claim would require *abort*.

The bank can use *commit* and $authorization_q$ to justify (to the consumer) marking the token spent and denying reuse or replacement. The consumer can demand this proof, requiring the bank to produce *commit*, which contains the merchandise encryption key, thus giving the consumer access to the goods. Note that in practice, assuming good faith, the consumer will acquire the key prior to this, and demand of proof will not be necessary.

The consumer can use *abort* and q to demand that the token be unlocked. This provides the bank with *abort*. Possession of *abort* assures the bank that it will not be subject to a claim that the transaction has completed, since such a claim would require *commit*.

Finally, the consumer can use *commit* and $goods_m$ to prove the contents of the delivered goods. The goods encryption key, the encrypted goods, and the description of goods (contained in the contract) have all been signed by M, and *commit* proves that the transaction completed. Some means for review of goods by an outside authority should be available to establish claims of incorrect or fraudulent goods delivery.

From a correctness standpoint, it is important to examine the use of combinations of signed, non-repudiable messages employed above. A given *commit* (or *abort*) is valid for only one combination of n, *expiration*, M, and L; it is considered to be compatible only with the messages which agree on those values. The one exception is that the certificate $goods_m$ does not mention L or *expiration*, and thus must agree only on n and M to be compatible. At each step, a party provides a signed message which can be used to prove that party's accountability with exactly the same range of *commit* messages as it would use in proofs of the preceding party's accountability. Thus, if a party is held accountable for the transaction, then it is provided with the non-repudiable messages that it needs to hold the previous party accountable as well.

5 Data Management

Our protocols rely heavily on the ability of the participants to hold each other accountable by maintaining records of each other's non-repudiable messages. We now discuss the record keeping required of the different participants in the protocol.

The consumer stores all data and messages received on all active tokens or transactions. This includes the token (Q^*), the signed contract and goods ($goods_m$), and finally the global commit (*commit*). The consumer can use $goods_m$ and *commit* to prove the contents of the goods and the contract. If the goods are satisfactory, then the consumer may discard all data on the transaction.

The merchant stores the bank's authorization ($authorization_b$) and then the global commit (*commit*). These are used to prove a completed transaction to the bank, which then issues a credit to the merchant. Once the bank has issued the credit, the merchant may discard all data on the transaction.

The transaction log stores the merchant's authorization ($authorization_m$) whenever it produces a global commit (*commit*). This information may be discarded after some delay following the transaction expiration (*expiration*). The delay should be long enough that denial of access to the log for that duration is extremely improbable. It is also important that the log not generate *abort* for any transaction with an *expiration* which has been exceeded by more than this delay period.

The bank must maintain a variety of transaction information to correctly manage the protocol. The bank must have a selection of *brands* of tokens. The brand of a token specifies the method used to create the token as well as the properties (e.g. denomination) associated with the token. Different brands of tokens are used to cover the range of desired token properties, particularly denominations and expiration dates. Since the bank knows the brand of the blinded token withdrawn by a consumer, the denominations and expiration dates should have a coarse granularity so that many tokens of each brand are issued. The bank must maintain a database of processing information for each brand of tokens it issues.

The database for a brand of tokens tracks the status of tokens of that brand. During the transaction phase, the bank receives an authorization ($authorization_q$) to commit a token to a transaction. This message is logged in the database, so that attempts at token reuse can be detected. Once the bank receives the the commit (*commit*) or abort (*abort*) for a transaction, it stores that as well. If the transaction completes, the bank should keep both $authorization_q$ and *commit* until some period following the expiration for the brand of token used. If the transaction aborts, the bank need keep only *abort*. In order to limit the time the bank must store *abort* records, transaction claims by the merchant should have a limited time of validity, perhaps some fixed period following *expiration*. This period should be sufficiently long to allow time for reasonable delays or for outside party conflict resolution. If the period is based on *expiration*, then the bank may decide to not authorize any transactions with excessively late *expiration* values.

Information / Party	Merchant	Consumer	Date	Amount	Item
Merchant	Full	None	Full	Full	Full
Consumer	Full	Full	Full	Full	Full
Bank	Full	None	Full	Full	None
Transaction Log	Full	None	Full	None	None
Law Enforcement w/warrant	Full	None	Full	Full	Full
Electronic Observer	Partial	None	Full	Partial	None

Table 1: Information Available with Anonymous Certified Delivery

6 Privacy

An important consideration in any transaction is what information is revealed about the participants and to whom. In this section, with the aid of a table, we detail what information is obtained by various agents.

Table 1 gives the types of information available to various parties in the style of [4]. The entries for the merchant, the consumer, the bank, and the transaction log are based on their original information plus any information received over the course of a transaction. The information for law enforcement with warrant assumes record-keeping on the part of the bank, and law enforcement's knowledge of the item is dependent on merchant records. The electronic observer's knowledge is based upon performing traffic analysis on the encrypted messages. In the basic protocol, the transaction log is publicly readable, and thus an observer can also obtain full information about the merchant's identity.

7 Trust

In this section we discuss the assumptions of trust necessary for our protocols. We then consider two modifications based on alternative trust assumptions.

While there are many places where a dishonest participant or saboteur could delay progress or prevent commitment (e.g. by disrupting a communication channel), there is only one location where a corrupt coalition may benefit illegitimately. For this reason, there is one trust assumption required by the protocol; the merchant must trust the log to record received messages. If the log, in collusion with the consumer, fails to produce *commit*, but simply passes k (which is contained in $authorization_m$) to the consumer, then the consumer will gain access to the goods while the merchant will not have *commit*, and thus will not be able to demand payment. In practice, if the time to *expiration* is sufficiently long and the log is accountable for responsiveness, this sort of fraud might be detected. Trusted outside observers could notice that the log is failing to respond in reasonable time and take some action.

This trust assumption against a log-consumer coalition is a reason for the existence of the transaction log as a separate entity. Before he commits to the transaction, the merchant knows the identity of the log, and therefore he need only commit if the specified log is trusted. In practice, the selection of the log might be decided in the initial negotiation between the consumer and merchant. If the merchant is assumed to trust the bank not to conspire with the consumer, the transaction protocol can be simplified by merging the bank and the log.

The second reason for a separate transaction log is the consumer's desire for timely access to the goods. From a practical standpoint, the consumer must trust that the log will not intentionally delay passing the key to the consumer. Although the key must eventually be revealed to the consumer for the bank to justify crediting the transaction, this would likely take place on a much larger time scale than would be desirable for key delivery. If the log is required to satisfy some responsiveness guarantees, then limited delays can be enforced with the assistance of a trusted outside party. If the consumer is assumed to trust the merchant to make timely delivery of the key (given that it must be delivered eventually), then the transaction protocol can be simplified by merging the merchant and the log.

8 Protocol Variations

The protocols presented in this paper form the groundwork for many variations which alter or extend their functionality. In this section we describe modifications to support key reuse, multiple token transactions, partial token spending, cryptographic time-stamps, a non-public transaction log, and full certified delivery.

8.1 Reusable Customer Keys

One variant of these protocols permits reuse of Q at the expense of allowing the bank to link repeated uses of Q: instead of blind signing $h(Q)$, the bank blind signs $h(Q, s)$, where s is some serial number chosen by the consumer. In this fashion, many tokens ($Q^* = Q, s, h(Q, s)^t$) can all specify the same key (Q). Only the bank sees Q (in P4), and so only the bank can link repeated uses of the same key to each other.

8.2 Multiple Token Transactions

To pay for items of arbitrary values, we may need to combine several tokens in a single transaction. In this case P2 must contain the various tokens, and be signed with all keys associated with those tokens. Let $\widehat{Q}^*$ be a list of tokens (possibly of different brands) $(Q_1, h(Q_1)^{t_1}), \ldots, (Q_m, h(Q_m)^{t_m})$ and let $\widehat{q}$ be the list of corresponding private halves $q_1, \ldots, q_m$. We extend the subscript notation to vectors to indicate signing the plaintext with each private key in the vector. The new P1 step would then be

P1. $C \rightarrow B \ \ (n, \mathit{expiration}, M, L, \widehat{Q}^*)_{\widehat{q}}$ — *authorization*$_q$

Multiple token transactions combine well with the use of one key for many tokens as discussed above, since this might reduce the number of signatures needed.

8.3 Partial Token Spending

The protocols may also be changed to support spending tokens in a check-like fashion. By including a particular value in P1, a consumer can authorize that only a part of a token's value is spent. The remaining value of the token may be exchanged for new tokens, or may be used in further purchases until all the value is used. Partial spending of tokens is compatible with both key reuse and multiple token transactions.

8.4 Cryptographic Time-Stamps

An important function of the log is to time-stamp the arrival of P4. The time-stamps used should include clock time information, since transaction authorization expiration will be in terms of real time. In order to reduce the trust that the parties must place in the log's honesty, cryptographic time-stamping [13] may also be employed. Cryptographic time-stamping will give the additional property that if the log is compromised, the log entries made prior to the time of compromise may still be trusted.

8.5 Encrypted Log Entries

In order to facilitate anonymous key acquisition by the consumer, the transaction log is publicly readable. While the logged message (*commit*) does not contain sensitive information, it might be used to determine the merchant's identity. Extra privacy could be supported by including a secret key (s) in the purchase messages. In fact, if n is required to be randomly selected and is sufficiently large, then n could be used as this secret key. The logged message would be encrypted using the secret key so that only the parties of the transaction could read (*commit*). To support efficient lookups, a function on known data could be used to generate indices for log entries (e.g. $E_s(M)$).

For even greater privacy, the log could be left unaware of the secret key and simply time-stamp, sign, and record any received messages (and their indices). This would require a modification of the *abort* message to indicate that no message with the given index was available at a specified time. Additionally, *expiration* should be left in plaintext so that the log can know not to publish messages with timestamps greater than their *expiration* values.

8.6 Two-Sided Certified Delivery

The last and most intricate variation on the protocols is the addition of support for two-way certified delivery. Our protocols provide one-sided certified delivery; only the consumer can prove what goods were delivered. If the burden of proof is expected to fall on the merchant, then the purchase protocol can be changed to provide full certified delivery at the cost of extra complexity. First, we introduce the notation $[M]_x$ to indicate the signature of M with key x without the plaintext, e.g., $h(M)^t \bmod N_t$ for RSA signatures. If we provide the merchant with Q and $[goods_m]_q$, then the merchant will be able to prove what goods were delivered to the holder of Q. The merchant must additionally be able to prove that the holder of Q is the consumer for whom the transaction was processed. Our purchase protocol for certified delivery follows:

CD1. $M \rightarrow C$ $(n, contract, E_k(goods))_m$ — $goods_m$

CD2. $C \rightarrow M$ $[goods_m]_q$ — $goods_q$

CD3. $C \rightarrow B$ $(n, expiration, M, L, Q^*)_q$ — $authorization_q$

CD4. $B \rightarrow M$ $(n, expiration, M, L, Q, \text{value})_b$ — $authorization_b$

CD5. $M \rightarrow L$ $(n, expiration, Q, k)_m$ — $authorization_m$ or

CD6. (a) L $((n, expiration, Q, k)_m)_l$ — $commit$
(b) L $((n, expiration, M, \text{failed})_l$ — $abort$

The new step, CD2, supplies the merchant with the signature by q of the goods description. The inclusion of Q in P3 enables the merchant to link Q with the payment to be received. The logging of Q in CD6a associates Q with the completed transaction. To increase the trustworthiness of this association in case of corruption by one or more parties, the cryptographic time-stamping variation described above should be employed. In variants where Q may be reused, the log entries should be encrypted to prevent unassociated parties from linking the repeated uses of Q.

9 Conclusion

In this paper, we presented protocols for achieving anonymous atomic transactions, answering an open question[16].

As stated in the introduction, these protocols are not being proposed for use in their current form. Both efficiency concerns and legal concerns — portions of the protocol may violate financial institution recordkeeping requirements on transactions over \$100 stipulated by the Money Laudering Act (12 USC §1829) in the US — must be addressed before such a protocol can be used. But an existence proof of an anonymous atomic protocol is an important step towards providing reliable, secure electronic commerce on the Internet, while maintaining individual privacy. Our variant protocol designs demonstrate the range of anonymous, ACID transaction available.

We hope that researchers and system designers in the electronic commerce community will further explore the technical feasibility of providing anonymous atomic electronic money transactions in real systems. We believe that these are fascinating technical issues and that in some contexts anonymous and reliable transactions will have important social value.

References

[1] Andrea J. Borr. Transaction monitoring in Encompass: Reliable distributed transaction processing. In *Proceedings of the Very Large Database Conference*, pages 155–165, September 1981.

[2] Stefan Brands. An efficient off-line electronic cash system based on the representation problem. Technical Report CS-R9323, Centrum voor Wiskunde en Informatica, 1993.

[3] E. Brickell, P. Gemmell, and D. Kravitz. Trustee-based tracing extensions to anonymous cash and the making of anonymous change. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 457–466, 1995.

[4] L. Jean Camp, Marvin Sirbu, and J. D. Tygar. Token and notational money in electronic commerce. In *Proceedings of the First USENIX Workshop in Electronic Commerce*, pages 1–12, July 1995.

[5] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In *Advances in Cryptology — CRYPTO '88 Proceedings*, pages 200–212. Springer-Verlag, 1990.

[6] David Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology: Crypto '82 Proceedings*, pages 188–293. Plenum Press, 1983.

[7] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, October 1985.

[8] Benjamin Cox. Maintaining prvacy in electronic transactions. Technical Report TR 1994-9, Carnegie Mellon University Information Networking Institute, Pittsburgh, PA, September 1994.

[9] C. J. Date. *An Introduction to Database Systems Volume 2.* Addison-Wesley, Reading, MA, 1983.

[10] J. Gray and A. Reuter. *Transaction Processing.* Morgan-Kauffman, 1993.

[11] James N. Gray. A transaction model. Technical Report RJ2895, IBM Research Laboratory, San Jose, California, August 1980.

[12] James N. Gray. The transaction concept: Virtues and limitations. In *Proceedings of the Very Large Database Conference*, pages 144–154, September 1981.

[13] Haber and Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3(2), 1991.

[14] Steven Low, Nicholas F. Maxemchuk, and Sanjoy Paul. Anonymous credit cards. Technical report, AT&T Bell Laboratories, 1993. Submitted to *IEEE Symposium on Security and Privacy*, 1993.

[15] Bruce Schneier. *Applied Cryptography, 2nd edition.* John Wiley and Sons, 1996.

[16] J. D. Tygar. Atomicity in electronic commerce. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 8–26, May 1996.

[17] Bennet S. Yee. *Using Secure Coprocessors.* PhD thesis, Carnegie Mellon University, 1994.