Unlinkable Serial Transactions

Paul F. Syverson[*] Stuart G. Stubblebine[**] David M. Goldschlag[*]

Abstract. We present a protocol for unlinkable serial transactions suitable for a variety of network-based subscription services. The protocol prevents the service from tracking the behavior of its customers while protecting the service vendor from abuse due to simultaneous or "cloned" usage from a single subscription. We present variants of the protocol supporting pay-per-use transactions within a subscription. We describe other applications including third-party subscription management, multivendor package sales, proof of group membership, and voter registration.

1 Introduction

This paper is motivated by an apparent conflict of interest concerning the privacy of information in an electronic exchange. Commercial service providers would like to be sure that they are paid for their services and protected from abuse due to simultaneous or "cloned" usage from a single subscription. To this end they have an interest in keeping a close eye on customer behavior. On the other hand customers have an interest in the privacy of their personal information, in particular the privacy of profiles of their commercial activity. One well known approach to this problem is to allow a customer to register with vendors under pseudonyms, one for each vendor [4]. By conducting transactions using anonymous electronic cash (e-cash) the customer's anonymity is maintained. But, the vendor is able to protect his interests by maintaining a profile on each of his anonymous customers.

In this paper we present effectively the opposite solution to this problem. The customer may be known to the vendor, but his behavior is untraceable. This would appear infeasible. If transactions cannot be linked to the customer, what is to keep him from abusing the service? For example, if someone fails to return a rented video, the video rental company would like at minimum to be sure that this person cannot rent any more videos. But, the company cannot do this if they cannot determine who the renter is.[1] We will present a protocol that makes transactions unlinkable but protects vendors from such abuses.

For the near future at least, a large part of the market on the Internet and in other electronic venues will rely on credit card based models (such as SET

[*] Center for High Assurance Computer Systems, Code 5543, Naval Research Laboratory, Washington DC 20375, USA. {*lastname*}@itd.nrl.navy.mil

[**] AT&T Labs–Research, Rm 2A-345A, 600 Mountain Ave., Murray Hill NJ 07974, USA. stubblebine@research.att.com

[1] In a pseudonym based scheme, such a customer could try to open an account under a new pseudonym, but there are mechanisms to make this difficult [5]. Thus, the interests of the vendor can be protected.

[16] or Cybercash [7] or simply sending credit card numbers over SSL). Applications of our protocol that require payment are not dependent on the payment mechanisms used. Thus, our protocol can be easily applied now but is equally amenable to use with e-cash. Even in an environment in which pseudonyms and anonymous e-cash are generally available, vendor profiles of customers (or their pseudonyms) might be undesirable because the customer's anonymity protection has a single point of failure. If the vendor is ever able to link a pseudonym to a customer, the entire profile immediately becomes linked to that customer. In our solution, if a customer is ever linked to a transaction, only his link to the one transaction is revealed. (This is somewhat analogous to the property of perfect forward secrecy in key establishment protocols.)

On what applications could our approach be used? Consider a subscription service for an on-line newspaper or encyclopedia. Customers might have an interest in keeping the searches they conduct private. At the same time, vendors would like to make it difficult for customers to transfer their ability to access the service. This will serve as our primary example.

We will also consider other applications. One example is pay-per-use service within a subscription (e.g., Lexis-Nexis or pay-per-view movies available to cable TV subscribers). Unlinkable serial transactions can also be used to provide multivendor packages as well as ongoing discounts. And, they can be used for anonymous proof of membership for applications having nothing directly to do with electronic commerce. Applications include proof of age and proof of residency. They can also be used to construct a simple voter registration protocol.

The paper is organized as follows. In section 2 we describe related work. Most of the basic mechanisms on which we rely come from work on e-cash; although, we are able to simplify some of those mechanisms for our purposes. We describe these and their relation to our work. We also rely on the assumption that communicating parties will not be identified by the communications medium, independent of the messages they send. Services that prevent this are discussed as well. In section 3 we will describe the basic protocol including set up, usage, and termination of a subscription. We also discuss recovery from broken connections. In section 4 we describe various applications of unlinkable serial transactions and associated protocol variants. In section 5 we present concluding remarks.

2 Related Work

2.1 Digital Cash

Digital cash, especially anonymous e-cash as presented by Chaum et al. [6], is characterized by several requirements [13]: independent of physical requirements, unforgeable and uncopyable, untraceable purchases, off-line, transferable, and subdividable. No known e-cash system has all of these properties, and certain properties, especially e-cash that can be divided into unlinkable change, tend to be computationally expensive.

E-cash can either be on-line or off-line. In an on-line scheme, before completing the transaction, the vendor can verify with a bank that the cash has not previously been spent. In an off-line scheme, double spending must be detectable later, and the identity of the double spender must then be revealed. Previously agreed upon penalties can then be applied that make double spending not cost effective.

Chaum's notion of *blinding* [5] is a fundamental technique used in anonymous e-cash and assigning pseudonyms. A bank customer may want a certain amount of e-cash from the bank, but may not trust the bank not to mark (and record) the e-cash in some way. One solution is for the bank to sign something for the customer that the bank cannot read, while the customer presents the bank with evidence that the bank is signing something legitimate.

Chaum's blinding depends on the commutativity of modular multiplication operations. Therefore, the customer can create an e-cash certificate, multiply it by a random number called a blinding factor. If the bank signs the blinded certificate, the customer can then divide out the blinding factor. The result is the unblinded certificate signed by the bank. But the bank does not know what it signed.

How can the customer assure the bank that the blinded certificate is legitimate? In Chaum's scheme, the customer presents the bank with many blinded certificates that differ in serial number, perhaps, but not in denomination. The bank chooses the one it will sign and asks the customer for the blinding factors of the others. If the randomly chosen certificates turn out to be legitimate when unblinded, the bank can have confidence that the remaining blinded certificate is legitimate too.

One on-line e-cash scheme is presented in [15]. To obtain an e-cash certificate that only he can use, a customer presents the bank with a hash of a random number. The bank signs an e-cash certificate linking that hash with a denomination. To use the e-cash, the customer reveals the random number to a vendor, who in turn takes the e-cash to a bank. Since hashes are one-way functions, it would be very hard for someone other than the customer to guess the secret that allows the e-cash to be spent. After the money is spent, the bank must record the hash to prevent it from being spent again. This scheme can be combined with blinding, to hide the actual e-cash certificate from the bank during withdrawal.

One off-line e-cash scheme is presented in [11]. There, the bank signs blinded certificates. To spend the e-cash, the customer must respond to a vendor's challenge. The response can be checked by inspecting the e-cash. Double spending is prevented because the challenge/response scheme is constructed so the combination of responses to two different challenges reveals the identity of the customer. As long as the customer does not double spend, his identity is protected. Nobody but the customer can generate responses, so the customer cannot be framed for double spending.

It may be the case that truly anonymous unlinkable e-cash enables criminal activity. Several key escrow or trustee-based systems [2] have been developed that can reveal identities to authorities who obtain proper authorizations.

Our notion of unlinkable certificates came from asking the following question: what else shares some of the features of digital cash? Unlinkable certificates share many of these features: they must preserve the user's anonymity and not be traceable, and they must protect the issuer and not be forgeable or copyable. Unlike e-cash, however, transferability is not desirable. We use hashing of random numbers and blinding in our development of unlinkable certificates. Our unlinkable certificates differ from Chaum's pseudonyms [5] which are an alternative to a universal identification system. Each pseudonym is supposed to identify its owner to some institution and not be linkable across different institutions. Unlinkable serial certificates are designed to be unlinkable both across institutions and across transactions within a single institution. In particular, we want the vendor to be unable to link transactions to a single customer, even if that customer had to identify himself initially (i.e., during the subscription process). At the same time, the vendor needs to be able to protect himself against customers that abuse his service.

Our blinding also differs from the usual approach. Typically some mechanism is necessary to assure either the issuing bank or receiving vendor that the certificate blindly signed by the issuer has the right form, i.e., that the customer has not tricked the signer into signing something inappropriate. We described Chaum's basic approach to doing this above. By moving relevant assurances to other parts of the protocols, we are able to eliminate the need for such verification. The result is a simplification of the blinding scheme.

2.2 Anonymity Services

How can a customer keep his private information private if communication channels reveal identities? For example, vendors having toll-free numbers can subscribe to services that reveal callers' phone numbers to the vendor thereby obviating any pseudonym the customer may be using. A similar service in the form of caller-id is now available to many private customers. If a communication channel implicitly reveals identities, how can customer's private information be protected?

The solution lies in separating identification from connections. The connection should not reveal information. Identifying information should be carried over the connection. (Of course, vendors and private parties are welcome to close connections that do not immediately provide sufficient identifying information.) On the Internet, depending upon one's environment and threat model, several solutions exist.

For e-mail, anonymous remailers can be used to forward mail through a service that promises not to reveal the sender's identity to the recipient. User's worried about traffic analysis can use Babel [12] or other Mixmaster [8] based remailers which forward messages through a series of Chaum mixes [4]. Each mix can identify only the previous and next mix, and never (both) the sender and recipient.

For Web browsing, the Anonymizer [1] provides a degree of protection. Web connections made through the Anonymizer are anonymized. By looking at connection information, packet headers, etc. the destination Web server can only identify that the connection came from (through) the Anonymizer.

Onion routing [17] provides anonymizing services for a variety of Internet services over connections that are resistant to traffic analysis. Like Babel, onion routing can be used for e-mail. Onion routing can also be used to hide Web browsing, remote logins, and file transfers. If the communicating parties have secure connections to endpoint onion routers, communication can be anonymous to both the network and observers, but the parties may reveal identifying information to each other. The goal of onion routing is anonymous connections, not anonymous communication. Other application independent systems that complicate traffic analysis in networks have been designed or proposed. In [9] a cryptographically layered structure similar to onions in onion routing is used to forward individual IP packets through a network, essentially building a connection for each packet in a connectionless service. In [14], mixes are used to make an ISDN system that hides the individual within a local switch originating or receiving a call.

3 Transaction Unlinkability

In this section we describe protocols that prevent linking of a client's transactions to each other. Consequently, they also cannot be linked to the client himself. We assume that the client has subscribed to a service with whom he will conduct these transactions and has provided adequate identifying and billing information (e.g., credit card numbers). The protocols make use of many basic e-cash primitives but are generally simpler than protocols using these primitives in their more common applications.

The basic protocol allows a customer to sign up for unlimited use of some subscription service for a period of time but prevents the service from determining when he has used the service or what he has accessed. At the same time, mechanisms are provided that make it difficult for the customer to share his subscription with others and leaves him vulnerable to detection and financial loss if he should do so. First we set out the requirements that such protocols should meet.

3.1 Requirements

Client Privacy Privacy of clients should be protected. Specifically, it should be difficult for a vendor or others to link the client to any particular requested transaction. It should also be difficult for the vendor to link any one transaction request with any other. (Thus, building a profile that might ultimately be tied to a client is difficult.)

Service Guarantee Clients should be assured that no one can steal from them the service for which they contracted, i.e., that vendors cannot be tricked into servicing invalid clients at their expense.

Fraud Prevention Vendors should be assured that they are not providing uncontracted services. Specifically, there should be no more active transaction requests for a service possible at any one time than the number of paid subscriptions at that time.

3.2 Basic Unlinkable Serial Protocol

The basic protocol has two phases, registration and certificate redemption, optionally followed by a termination phase. The goal of registration is to issue credentials to a new subscriber. The new subscriber, C, presents sufficient identifying and payment information to the vendor, V. The vendor returns a single blinded certificate, which authorizes the client to later execute a single transaction with that service.

In the certificate redemption phase, clients spend a certificate and execute a transaction. At the end of the certificate redemption phase, the vendor issues the client another blinded certificate. The vendor cannot link the new certificate to the spent one, so he cannot use it to link transactions to one another.

We assume that the customer has an associated identifier C for each account, whether or not his identity is actually known by the vendor. (He may in fact have different identifiers for different accounts.) We use square braces to indicate message authentication and curly braces to indicate message confidentiality. Thus, '$[X]_K$' might refer to data X signed with key K or a keyed hash of X using K. '$\{X\}_K$' refers to X encrypted with key K. For our purposes both of these are used to refer to mechanisms that also provide message integrity. We use over-lining to indicate blinding: e.g., '$\overline{X}$' refers to the result of blinding X, for use with the appropriate signature key.

3.3 Registration

$$\begin{array}{lll} \text{Message 1} & C \rightarrow V: & \{Payment, K_{audit}, CreditAuth, K_{CV}\}_V, \\ & & \qquad [Request\ for\ certificate\ of\ type\ S, C, \overline{h(N_1)}]_{K_{CV}} \\ \text{Message 2} & V \rightarrow C: & [\overline{h(N_1)}]_S \text{ (OR } [Not\ approved]_{K_{CV}}) \end{array}$$

The signature key in message 2 is the vendor's signature key for service S and is only used to sign blinded hashes. A signed hash is a certificate. The service key is also subject to periodic renewal. Service keys have published expiration times. All certificates should be used or exchanged by that time. We will see that there is no need to verify the structure of the blinded hashed nonce. If the client substitutes anything inappropriate the result can only be an invalid certificate. In message 1, the *CreditAuth* is a credit authorization which is returned by V when the subscription is terminated.[2] To receive *CreditAuth* the client must produce the secret K_{audit}. *CreditAuth* can be held by V in the event C fails an audit. (Audits will be described below.) The decision as to whether V actually draws against this credit is a policy decision and is outside the scope of this paper.

The vendor must remember this sequence of messages, in case message 2 was not received by the client. (See section 3.8.) For this registration protocol, the

[2] In other protocol variants, *CreditAuth* can be a form of deposit. However, a traditional deposit is sometimes undesirable since money is held for the entire term of the subscription.

service should consider message 2 to have been received after some period of time. For (space) efficiency, an acknowledgement message may be added:

Message 3 $\quad C \rightarrow V: \quad [Ack]_{K_{CV}}$

A customer may wish to make use of his subscription from multiple machines, e.g., a base machine at his home or office and a laptop machine used when traveling. It may be considered too much of an inconvenience to require the customer to transport the current unspent certificate for each of his subscriptions to his next likely platform of use. The vendor may therefore allow the customer to obtain a number of initial certificates, possibly at no additional fee or for a nominal charge. Similarly, the customer might be allowed to add an initial certificate during his subscription if he begins using a new machine. The vendor will need to decide which policy best meets his needs.

K_{CV} is used to link protocol messages to one another. This becomes even more important when certificates are redeemed for transactions. We will discuss further assumptions and requirements regarding this linking after presenting the certificate redemption protocol.

3.4 Certificate Redemption

When the customer wants to make use of the service, he conducts a certificate redemption protocol with V. Certificate redemption consists of certificate spending, transaction execution, and certificate renewal.

Message 1 $\quad C \rightarrow V: \quad \{[h(N_i)]_S, N_i, K_{CV}\}_V,$
$\qquad [\mathit{Request\ for\ transaction\ of\ type}\ S, \overline{h(N_{i+1})}]_{K_{CV}}$

Message 2 $\quad V \rightarrow C: \quad [\mathit{Approved}]_{K_{CV}}$
$\qquad$ (OR $[\mathit{Not\ approved}]_{K_{CV}}$ OR $[\mathit{Audit}]_{K_{CV}}$)

Message 3 $\quad C \leftrightarrow V: \quad [\text{Transaction}]_{K_{CV}}$

Message 4 $\quad V \rightarrow C: \quad [\overline{h(N_{i+1})}]_S$

The transaction, message 3, is only done if message 2 was $[\mathit{Approved}]_{K_{CV}}$. The other two possibilities are discussed in the next sections. We delay the release of the new certificate $[\overline{h(N_{i+1})}]_S$ until the transaction ends, to prevent the client from beginning a new certificate redemption protocol before the current one completes. If the new certificate were released before the transaction, a subscriber could run his own subscription server which would proxy transactions for *his* clients.

K_{CV} is a key that is used to protect the integrity of the session; C should choose it to be unique for each session. If K_{CV} should be compromised and then used in a later session, an attacker could create her own second field in the first message. By so doing, she could hijack the subscription.

Uniqueness of K_{CV} is thus important to honest customers. But, session integrity is important to the vendor as well. The vendor would like to be sure that transaction queries are only processed in connection with a legitimate certificate renewal. Unfortunately, K_{CV} may not be enough by itself to guarantee integrity of a protocol session. One or more customers might intentionally reuse the same session key and share it with others. Anyone who has this key could then submit queries integrity protected by it. As long as such a query is submitted during an active legitimate session for which it is the session key, there is nothing in the protocol that distinguishes this query from legitimate queries. This would allow wide sharing of subscriptions by effectively bypassing certificate spending. Other aspects of protocol implementation might prevent this. But, to be explicit, we will assume that uses of K_{CV} are somehow rendered serial within a protocol run. For example, K_{CV} might be used in the protocol in a stream cipher. Alternatively, K_{CV} might be used as a secret nonce that is hashed with plaintext. The plaintext and hash are sent in each message. Each time a message is sent the nonce could be incremented. If something is done to make each use of K_{CV} in a protocol session unique and tied to previous uses within that run, then sharing of subscriptions by this method becomes at least as inconvenient as sharing them by passing the unspent certificate around. We make this same assumption for all protocols mentioned in this paper that use a session key to protect the integrity of the session.

As in the registration protocol, the vendor must remember the messages sent in this protocol (except for the transaction messages) in case the client never received the new (blinded) certificate. For efficiency, an acknowledgement message may be added:

Message 5 $\quad C \rightarrow V: \quad [Ack]_{K_{CV}}$

3.5 Not approved

If the response in message 2 is *Not approved*, then the protocol terminates. The response to a request for service might be *Not approved* for a number of reasons. These include that the certificate has been spent already, the nonce does not match the submitted certificate, and the certificate is not valid for the service requested. Alternatively, the certificate submitted might use an expired key. If the client is a valid subscriber who never received an initial certificate for the current key, this should be reflected in the vendor's records. The client can then get an initial certificate in the usual manner. Off-line appeal will be necessary for clients who feel they have been refused a legitimate transaction request. We have designed these protocols under the assumption that appeals will be automatically decided in favor of the client, as long as the client has not appealed too many times.

3.6 Audit

If the response is *Audit*, then a special audit occurs in which C must present some proof that he is a valid subscriber within a short period of time. In particular, C

must prove knowledge of K_{audit}, which was sent to V during registration. If this is satisfactory, a new certificate is issued. If it is not satisfactory or if C does not comply, then the protocol terminates, and the certificate is logged along with a note that it was used during a failed audit. In either case, no transaction takes place so audited customers are not linked to specific transaction requests. The main purpose of audits here is to serve as a secondary deterrent to sharing a subscription with a nonsubscriber. (The primary deterrent is the inconvenience of passing the certificate back and forth between those sharing as compared with the cost of obtaining another subscription.) We will see that if anyone can demonstrate knowledge of K_{audit} and provides a valid certificate, then he can terminate the corresponding subscription and the vendor will transfer *CreditAuth* to him. Thus, C will not want to share K_{audit} with anyone whom he does not trust not to redeem the *CreditAuth*. If a customer is ever caught during an audit having given away his certificate but not his K_{audit}, he effectively forfeits his subscription (and *CreditAuth*). This is because that certificate can never be used again, and no new certificate is issued to continue the subscription. Off-line appeal mechanisms may again be available for customers who, for example, lose certificates or secret nonces.

The audit protocol is as follows:

Message 1 $C \rightarrow V:$ $\{[h(N_i)]_S, N_i, K_{CV}\}_V,$
$[\mathit{Request\ for\ transaction\ of\ type\ S}, \overline{h(N_{i+1})}]_{K_{CV}}$

Message 2 $V \rightarrow C:$ $[\mathit{Audit}]_{K_{CV}}$

Message 3 $C \rightarrow V:$ $\{C, K_{audit}\}_{K_{CV}}$

Message 4 $V \rightarrow C:$ $[\overline{h(N_{i+1})}]_S$ (OR $[\mathit{Not\ approved}]_{K_{CV}}$)

Similarly to the basic certificate redemption protocol, if message 4 is $\{\mathit{Not\ approved}\}_{CV}$, then the protocol terminates. Unlike the basic certificate redemption protocol there is no transaction phase. So, there is no direct link between any identifying information revealed in the audit and any particular transaction. However, by exercising the audit check frequently or at strategic times, the vendor can learn both the client's usage frequency and patterns. This might allow the vendor to correlate later transactions (and possibly earlier transactions) with the particular client. The client might counter this limitation by employing a masking scheme on top of the basic protocol. However, this can considerably increase the load on the subscription service. Clients might also counter such vendor analysis by delaying ordinary transaction requests for a random amount of time following an audit. This places no extra burden on the subscription service but may cause customers inconvenience substantially beyond that of audits themselves. Since audits are a secondary deterrent to abuse, they might be conducted infrequently. The tradeoffs between threats to anonymity and the deterrence effect on subscription sharing are difficult to assess a priori. Thus, exactly how frequent to make audits is currently difficult to say.

The service must remember the sequence of messages in any run of this protocol in case of a broken connection. The messages may be remembered until the associated key expires, or until some amount of time has elapsed, after which the new certificate is assumed to have been received. For efficiency, an acknowledgement message may be added:

Message 5 $\quad C \rightarrow V: \quad [Ack]_{K_{CV}}$

We will see in the next section why customers will want to protect K_{audit}. In message 3 of the audit protocol we explicitly use K_{CV} as an encryption key. In other cases, we encrypted for the vendor using V. (In practice, the symmetric key K_{CV} would typically be used in favor of the computationally expensive public key V.) However, it is essential that V *not* be used in message 3, since that would allow a subscriber to share his subscription, and produce responses to audit challenges without revealing his secret K_{audit} to those he shared with.

3.7 Terminating a Subscription

Client initiated termination of a subscription is a variant of certificate redemption, however, it does not trigger an audit. Termination requires the client to prove it knows K_{audit} and has an unspent certificate. Termination has the effect of passing the *CreditAuth* to the subscriber. V passes *CreditAuth* but one time.

Message 1 $\quad C \rightarrow V: \quad \{[h(N_i)]_S, N_i, K_{audit}, K_{CV}\}_V,$
$\qquad\qquad\qquad\qquad [\textit{Request for transaction of type (S Termination),C}]_{K_{CV}}$
Message 2 $\quad V \rightarrow C: \quad \{\textit{CreditAuth, Refund}\}_{K_{CV}}$ (OR $[\textit{Not approved}]_{K_{CV}}$)

Refunds may be prorated based on the vendor's policy for early termination. Should the subscription include multiple chains of certificates (e.g., for a workstation and a laptop) there should be one *CreditAuth* per chain.

In message 2, we encrypt using K_{CV} since we do not require that the client possess a private key.

As before, an acknowledgement message may be added for efficiency:

Message 3 $\quad C \rightarrow V: \quad [Ack]_{K_{CV}}$

3.8 Recovering from Broken Connections

Protocols that break before the vendor receives the acknowledgement must be replayed in their entirety (except for the actual transaction which is always skipped), with the same session key, nonce, and blinding factor. The protocols are designed not to release any new information when replayed.

Broken protocols are considered automatically acknowledged after some period of time (i.e., the customer has that much time to recover from a broken connection). After that period of time, they can no longer be replayed. This is not crucial for the redemption protocol, but is crucial for the registration protocol. After that period of time, the subscription may be charged for.

We will consider connection breaks occurring from the end of the protocol to the beginning. If a connection breaks after a new certificate has been acknowledged (message 5 in the Certificate Redemption protocol), the client can simply initiate a new transaction with the new certificate. If a connection breaks after C receives message 4 but before V receives message 5, the client can again simply initiate a new transaction.

Before this point in the protocol the client will not yet have received a new certificate. So, recovering from any connection breaks that occur prior to this point in the protocol involve replaying the protocol. The vendor should keep a record of each protocol run until he receives the acknowledgement in message 5. Upon replay, the client presents the same sequence of messages. The vendor will identify the presented certificate as spent, and consult its recovery database. If the protocol is recoverable (i.e., has not yet been acknowledged), the vendor returns the stored response.

If the response in message 2 is $Audit$, V should keep a record of the protocol run even if C properly identifies himself upon reestablishing the connection. It may be that a cheater broke the connection and then quickly notified the legitimate client of the audit. If some client breaks an audit protocol repeatedly a vendor may become suspicious and may decide not renew his certificate.

Notice that the customer need never identify himself when a broken connection occurs (unless an audit had already been stipulated by the vendor). Thus, he need not worry about being associated with a given transaction.

Another kind of failure that affects our system is disk crash or other media failure. It is unrealistic and unreasonable to expect customers to backup copies of subscription information every time they redeem a certificate. (It is often unrealistic to expect customers to make backups at all.) Therefore, customers must be allowed to reinitialize a subscription after a disk crash. How often individuals will be allowed to reinitialize over the course of a subscription is a policy decision for individual vendors. Another option is to provide customers with (distinct) backup initial certificates at registration, just as they may obtain initial certificates for multiple machines. This allows them to recover from a disk crash without re-registering (assuming they have kept backups separately); however, it does provide additional subscription chains for the cost of one subscription.

3.9 Service Key Management

For unlinkable protocols to work, it is important that service keys not be "closely" associated with clients. For example, we do not want the vendor to be able to uniquely associate a service key with each client, which would enable the vendor to associate transactions with clients.

Committing to Service Keys A straightforward technique to overcome this potential vulnerability requires the vendor to publicly commit to all public authorization keys. This can be achieved by publishing information, at regular intervals, at a unique location "well known" to all potential clients of the service. An example publication format for each service consists of the service type, expiration time, and signature confirmation key for signatures associated with this service.

Subscription Termination Other than as a general security precaution, the primary reason to change service keys is to facilitate expiration of subscriptions. When keys expire, our only current mechanism is to have clients obtain new certificates just as they did when signing up for a service initially. Service expiration can be structured in several different ways, each with advantages and disadvantages. We will present some of these and briefly mention some of the tradeoffs. Which is most acceptable will depend on particular aspects of application and context. For the purposes of discussion let us assume that the standard period of subscription is one year divided into months.

Subscription Expiry One option is to have annualized keys that start each month. In other words, there are twelve valid service keys for the same service at all times. This is convenient for the customer and similar to existing subscription mechanisms; however, it partitions those using a service into twelve groups, reducing the anonymity of customers accordingly. This may or may not be a problem. If subscriptions are annualized to quarters this reduces the threat to anonymity, but this might still be unacceptable. And, it reduces customer flexibility about when subscriptions can begin.

An alternative is to have monthly keys good for all subscribers. Subscribers obtain twelve seed certificates when they subscribe, one for use in each month of the succeeding year. This does not reduce anonymity as the last option did. On the other hand, it requires that customers keep track of the multiple certificates and requires issuing certificates well in advance of their period of eligibility. From the vendor's perspective, the threat of audit becomes much reduced since a cheater will lose at most the current month's certificate. Relatedly, it is that much easier to share a subscription—at least by monthly pieces. Thus, the inconvenience deterrent is reduced slightly as well.

Another option is to have all subscriptions end in the same month. Someone subscribing at other than the beginning of the fiscal year would pay a prorated amount for his subscription. This avoids reductions in anonymity associated with monthly annualized keys. It also avoids the reduced deterrence to cheating associated with monthly keys. But, it reduces customer flexibility in choosing the ending of the subscription. Another disadvantage to this approach is that subscription renewal is now all concentrated at one point in the year, creating extremely unbalanced load on the part of the system handling sign up and renewal. This would probably remain true even if renewing customers were allowed to renew in advance. It could be diminished by splitting the year in half or even further. This creates the partitioning reduction in anonymity already mentioned.

Early Termination of a Subscription Terminating a subscription early requires proving that the user is a particular subscriber and spending a valid certificate. He will not get a new one; so, there is is no way for him to continue using the service. Notice that early termination can even be customized, for example, so that it is available only to customer's who have already subscribed for at least a year. (Recall that a customer reveals his identity, or pseudonym, when he terminates early.) Prorating refunds for terminated subscriptions removes one of the disadvantages of the third option for subscription expiration described above.

We have been describing subscriber termination of a subscription. Vendor termination of a particular subscriber or group is far more difficult. (It may also be less important.) In our current approach the only way to terminate a subscriber is to change the service key(s) for the remainder of his subscription and require everyone else to reinitialize their certificates with the new key. This creates tremendous expense and inconvenience equivalent to what would be necessary if a service key were compromised.

3.10 Discussion

The protocols presented thus far have limitations in protecting against defrauding the vendor by organizing a service to share subscriptions. It seems doubtful that a practical solution exists to fully protect against this attack, given our goal of unlinkable transactions. For example, subscriptions may be shared if the subscriber runs a subscription proxy server. But, this makes sharing a centralized activity, with the attendant complexity of running a new business. Such a business has the overhead and complexity of marketing, advertising, and maintaining service reliability. Perhaps more importantly, it has the potential disadvantage of being a focus for legal attention. Finally, the vendor can take action against the particular shared account if it shows up frequently in an audit.

If the registered subscriber is not running a subscription proxy service but is lending his unspent certificate, what can be done to make sharing more centralized? In addition to the mechanisms already in place, the key is to require intimate contact between the lender and the borrower. Sharing is inherently risky to the lender because the borrower may never return the subscription. Thus the lender should require a deposit. However, requiring a deposit or charging for fraudulent activity has historically been a key element to detecting and limiting fraud.

Another approach that forces lending to be centralized, which complements the approach just presented, is to design the protocol so the borrower must contact the lender on every transaction (if the lender does not want to share all of his secrets). Currently, a borrower only needs to contact the lender when audited (to get K_{audit}). Alternatively, one could modify the protocol to require K_{audit} to be indirectly present in the first message of every run of the certificate redemption protocol. For example, the client could send a hash of the spent certificate, K_{audit} and a random number in message 1, the later two of which must be revealed in the event of audit.

4 Applications of Unlinkable Serial Transactions

Until now we have been focused on basic subscription services as the application of unlinkable serial transactions. We now explore both expansions of the basic subscription application and other applications as well. We will simply describe these applications without giving full details on how to adapt the unlinkable serial transactions for them. Generally, it will be straightforward to see how to do so.

4.1 Pay-per-use Within a Subscription

Certain transactions may require extra payment by a subscriber. Next, we describe a means to allow pay-per-use within a subscription. The vendor becomes a mint for simple, single denomination, digital tokens. The digital tokens are to digital cash roughly as tokens in a game arcade are to coins. The vendor may bill for these tokens by credit card, or some other mechanism.

During the transaction phase (message 3 in the certificate redemption protocol), the client spends previously purchased tokens. How do we guarantee that the client pays the vendor for the pay-per-use transaction? Either the vendor never releases the new blinded certificate (message 4) unless he is payed or we assume some protocol for fair exchange [10, 3]. The latter choice properly partitions responsibility without complicating recovery.

There are alternatives to this protocol. For example, certificates could include a credit balance, which must be periodically paid. Payment would be made as a transaction. There is no harm in this transaction identifying the customer because it is only for payment purposes. The main limitation on this approach is that the credit balance is monotonically increasing. This may allow the vendor to link transactions and even to tie them to particular customers.

4.2 Third-Party Subscription Management

Vendors may be interested in making available the anonymity afforded by our approach but may be less enthusiastic about the necessary overhead of maintaining a subscription, e.g., keeping track of spent certificates. Along with the ordinary overhead of maintaining subscriptions, handling billing, etc., vendors may choose to hire out the management of subscriptions. It is straightforward to have the vendor simply forward transaction requests to a subscription management service, which then negotiates the business (certificate management) phase of the protocol with the customer. Once this is completed, the transaction phase can proceed between the vendor and the customer as usual.

4.3 Multivendor Packages and Discount Services

For multivendor packages one can purchase what is effectively a book of coupons good at a variety of individual vendors. The way a coupon book would work is that vendors will authorize the package vendor to issue certificates for their services. Customers then engage in a protocol to obtain the basic certificates.

If the coupons in the book are meant to be transferable, there is nothing more to the protocol. If, however, they are not, we must add a serial unlinkable feature to make sharing more cumbersome. In this case, when a customer submits a certificate for a service he must also submit a package certificate. The package certificate must be updated as in the basic protocol. Service certificates are not to be updated: they can only be redeemed once. Vendors could all be authorized with the necessary key to update the package certificate. Alternatively, the processing of the certificates could be handled by the package issuer as in the third-party application of unlinkable serial transactions just given. Notice that individual vendors need not be capable themselves of producing coupons for their own services. It is enough that they can confirm the signatures associated with their services.

Package books such as just described often offer discounts over vendors' basic rates as a sales incentive. Another form of discount is one that is made available to members of some group. Unlinkable serial transactions are useful for allowing someone to demonstrate such membership without revealing his or her identity. Depending on the application, the various vendors offering discounts can sign new certificates or signing can be reserved for some central membership service in association with any request for discount at a vendor. The latter case is again similar to the third-party application above.

4.4 Membership and Voting

The example just mentioned shows that the basic idea of unlinkable serial transactions can have application outside of commercial concerns. Specifically it should be useful for any application for which membership in some group must be shown, and where the inconvenience of sharing a serial certificate and the risk of audit outweighs the advantages of spoofing group membership. These might include some applications requiring proof of age or residency.

As another example, consider a voter registration certificate. At voting time, the voter spends his certificate, is issued a new certificate, and votes. The new certificate is signed by a key that becomes valid after the current voting period expires, so voters cannot vote twice. In this case, there is no possibility of sharing the certificate for a single election. If there is concern that formerly eligible voters continue to vote once their eligibility has expired, certificate keys could be subject to occasional expiry between elections. Ineligible voters would then be eliminated since they would be unable to register for new seed certificates.

5 Conclusion

In this paper we have presented a protocol that can be used for unlinkable serial transactions. The protocol can be used in several types of commercial services, including unlimited use subscriptions and those incorporating some kind of pay-per-use transaction. Unlinkable serial transactions can also be used for multivendor packages and discount services. And, they can be used for non-commercial applications such as voter registration and proof of group membership. Although individuals are anonymous during each unlinkable serial transaction, they can be challenged to produce identification to prevent various kinds of fraud.

Our approach relies on anonymous communication: there is no sense in using anonymous tokens, pseudonyms, etc., if identities are revealed by the communications channel. For Web based commerce, the Anonymizer hides the identity of clients. Onion routing also provides anonymity, but in addition protects against traffic analysis and hides anonymity even if some of the nodes in the anonymity service are compromised.

In this paper we have described means to prevent profiling by vendors. But, profiles may be beneficial to both the customer and vendor, e.g., for marketing purposes. Indeed, services such as Netangels and Firefly are available that build customer profiles for this purpose but promise to protect customer privacy. It might be complicated to incorporate such trusted intermediaries with the protocols we have presented. But, decentralizing may ultimately provide better assurance to customers. Profiles can be collected locally at a user's workstation. This lets individuals control their own profiles. An individual could contact a marketer through an anonymous connection (cf. Section 2.2) and request advertisements suited to his profile. Once he closes the connection the marketer can no longer contact him.

Our approach is based on primitives supporting e-cash but is designed to function in a credit card type commercial infrastructure as well. By manipulating what must be trusted and by whom, as compared with their more common applications, we are also able to simplify the use of such primitives in our protocols.

References

1. Community ConneXion, Inc., http://www.anonymizer.com.
2. E. Brickell, P. Gemmell, and D. Kravitz, "Trustee-based Tracing Extensions to Anonymous Cash and the Making of Anonymous Change", Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 457–466, San Francisco, California, 22-24 January 1995.
3. L. J. Camp, M. Harkavey, B. Yee, J. D. Tygar, "Anonymous Atomic Transactions", Second USENIX Workshop on Electronic Commerce, 1996.
4. D. Chaum, "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms", CACM 24, 2, Feb. 1981, pp. 84–88.
5. D. Chaum, "Security without Identification: Transaction Systems to Make Big Brother Obsolete", CACM (28,10), October 1985, pp. 1030–1044.
6. D. Chaum, A. Fiat, and M. Naor, "Untraceable Electronic Cash", CRYPTO88, pp. 319–327.
7. http://www.cybercash.com. CyberCash is a trademark of CyberCash, Inc.

8. L. Cottrell, “Mixmaster and Remailer Attacks”, http://obscura.obscura.com/~loki/remailer/remailer-essay.html
9. A. Fasbender, D. Kesdogan, O. Kubitz. *Variable and Scalable Security: Protection of Location Information in Mobile IP,* 46th IEEE Vehicular Technology Society Conference, Atlanta, March 1996.
10. M. Franklin and M. Reiter, “Fair Exchange with a Semi-Trusted Third Party”, Fourth ACM Conference on Computer and Communications Security, Zurich, April 1997.
11. M. Franklin and M. Yung, “Towards Provably Secure Efficient Electronic Cash”, Columbia University CS Technical Report, TR CUCS-018-92, 1992.
12. C. Gülcü and G. Tsudik, “Mixing Email with *Babel*”, 1996 Symposium on Network and Distributed System Security, San Diego, February 1996.
13. T. Okamoto and K. Ohta, “Universal Electronic Cash”, CRYPTO91, pp. 324–337.
14. A. Pfitzmann, B. Pfitzmann, and M. Waidner. *ISDN-Mixes: Untraceable Communication with Very Small Bandwidth Overhead,* GI/ITG Conference: Communication in Distributed Systems, Mannheim Feb, 1991, Informatik-Fachberichte 267, Springer-Verlag, Heildelberg 1991, pages 451-463.
15. D. Simon, “Anonymous Communication and Anonymous Cash”, CRYPTO96, pp. 61–73.
16. Secure Electronic Transaction (SET) Specification. August 1, 1996.
17. P. Syverson, D. Goldschlag, and M. Reed. Anonymous Connections and Onion Routing, to appear *Proceedings of the Symposium on Security and Privacy*, Oakland, CA, May 1997.