Improvements in Probabilistic Micropayment Schemes

Daphne Lucas
98128241

Major Research Project

August 23, 2004

Abstract

This paper reviews the desirable properties and security features of electronic commerce systems and the problems that are unique to electronic cash. We will then examine two older micropayment systems, describing the desirable properties that they achieve. Finally, we will present a new micropayment system for electronic commerce introduced by Silvio Micali and Ronald Rivest, including the digital encryption technologies employed, which improves upon past systems. This last system (used in Peppercoin Payment Service) uses a probabilistic protocol to efficiently process small transactions, choosing only a certain number of cryptographically secure payments to process which reduces bank processing costs by several orders of magnitude.

1. Introduction

Micropayment systems do not have a very long history, but their track record has been very consistent: they do not work. Almost every micropayment system that has been implemented in attempt to transfer small amounts from person to person has failed, usually due to expensive computations or administration. In fact, 18 of 19 systems analyzed by Goldie-Scot failed [10], begging the question, "does anyone really need micropayments?" To achieve user acceptance, micropayment systems must be very easy to use, and to be implemented by vendors, they must be very efficient, guaranteeing that the cost of the system will not exceed the value of the payments [9]. Some people argue that subscriptions are preferred by vendors because they can sell bundles of merchandise, generating more income. Others argue that micropayments and subscriptions can co-exist, especially when the marginal cost for the production of information goods is zero [10]. In the newspaper industry, pay-per-use sales amount to approximately 31% of total revenue, and currently internet pay-per-use is at 14% and growing. This leads one to believe that even though subscriptions will be the dominant means of payment, micropayment schemes can help increase the cash flow with pay-per-use revenue.

Having established that micropayments will in fact play a role in the future of digital transactions, the remainder of this paper will discuss the history and the future of some specific schemes. Section 2 briefly reviews the history of electronic commerce, including the desirable properties and security features of electronic commerce systems and the problems that are unique to electronic cash. Section 3 presents the main goals of micropayment systems and a brief overview of previous schemes and techniques. Section 4 describes the details of Rivest and

Shamir's *Payword* scheme, including the efficiency of the scheme, the relationships between the users, vendors, and banks, and the problems which render it impractical for real world use. Section 5 describes the specific details of, and the problems associated, with Rivest's *Lottery* Scheme: the first system in which the bank does not process each payment from the user. Section 6 explains the details of the three new payment schemes that fix the problems associated with *Payword* and the *Lottery* schemes. The final scheme, MR3, which has been successfully implemented in the *Peppercoin Payment Service* is discussed in Section 7. In conclusion, Section 8 briefly discusses some current payment schemes that seem promising for micropayments.

2. Electronic Commerce: a Brief Overview

As world becomes more and more dependent on network communications to access, store, and distribute information, secure and efficient electronic commerce systems are becoming a necessity. We use the term electronic commerce (or e-commerce) to describe any financial transaction involving the electronic transmission of information, where the packets of information that are transmitted are called electronic tokens [2]. An electronic payment is a type of e-commerce where a series of transactions leads to a payment (which can involve tokens issued by a third party). Examples of electronic payment systems include digital cheques, debit cards, credit cards, and stored value cards. In this system, we call the consumer/user the *payer*, the vendor/merchant the *payee*, and the Bank is the financial institution where both payer and payee hold accounts. We will now discuss the security goals, the desirable properties, and the problems of electronic payment systems.

Security Goals

The security goals of electronic payment systems are: privacy, authenticity (including both user identification and message integrity), and non-repudiation [2]. These goals are achieved through an authentication infrastructure where privacy is achieved by enciphering each message with a public key. In order to achieve genuine privacy, we need true anonymity ensuring that a user's history of purchases is not available to any other party (i.e. bank or credit card company). True anonymity would include *payer anonymity* during payment, and also *payment untraceability* preventing the Bank from detecting who paid for a particular transaction. One way to achieve genuine privacy is through value cards where the user transfers money from

their account to their card, and then from their card to a particular vendor. Authentication (including user identification, message integrity, and non-repudiation) is achieved through key management, where a certificate authority or trusted third party is used to confirm the user's identification. This authority issues each user of this system (payer, payee, and bank) an identity certificate which they will use to prove their identity to others with whom they wish to do business. This certificate also allows each system user to set up private keys between users in a secure and authenticated way [2]. Non-repudiation is achieved through digital signatures using the public/private key pairs. The security of the payment system depends on the security of the authentication infrastructure, which is usually separate from the payment system itself. All of these security goals can be achieved through digital signatures based on a public key infrastructure which includes one-way collision-free hash functions and public/private key pairs signed by a certificate authority or trusted third party.

Desirable Properties

We would also like to model properties of paper cash in the electronic cash system. For example, paper cash is portable, recognizable (as legal tender), transferable (one can respend it without interaction of the bank), untraceable, anonymous, and divisible (can make change) [2]. Electronic cash systems focus primarily on preserving the untraceability and user anonymity properties. A payment system is said to be transferable if it allows at least one transfer of coin or token before interaction with the Bank. There is not very much research in this area, and so far all electronic payment systems have been non-transferable. One major problem associated with transferability is that coins *must* grow in size with each transfer in order to keep information about each user who has spent the coin. Thus, due to space limitations, we cannot have a system with an unlimited number of transfers as in the paper cash system. Another major problem is that until the token is deposited in the Bank, the Bank only knows the identification of the person who originally withdrew the money and would therefore need the co-operation of successive spenders in order to reconstruct the entire chain of transactions. This complication adds delay to the reconstruction, and by the time the chain of events is reconstructed, it may be too late to catch the dishonest user who has double spent the coin or used a forged coin. The schemes discussed in this paper will not be untraceable, but all of them could be modified to be untraceable (by adding another party) as we will describe in Section 3. Divisibility is needed in any payment system because it is just as unlikely that a user will have exact paper change to pay

for a transaction as it is that a user will have a token that is the exact amount of the transaction. Users do not want to carry around a large number of tokens to add up to the correct change for the same reasons that we do not want to carry around large amounts of paper cash: loss of interest and fear of it being stolen [2]. Divisibility of tokens/coins allows users to make payments without the need of storing a large number of tokens of different denominations. There have been off-line divisible electronic cash schemes proposed, but they result in longer transaction time and larger storage requirements. One way to implement this kind of scheme is to use binary trees where nodes have monetary values and a path in a tree corresponds to a transaction. Thus, in order to prevent double spending, no path can use the same node as another path. Also, once a node has been used, its' descendants and ancestors cannot be used. The identity of a payer is not revealed if all their payments are valid, but once a payer spends the same token/coin more than once, all purchases made with this coin will be revealed (including valid payments). Divisible payment systems are definitely more complicated to implement, but they are possible without forfeiting the prevention of double spending and untraceability. All of the schemes discussed in this paper (Payword, Lottery, and MR3) are divisible in their own way, either through tokens or because the merchant has the credit card number of the user and does not need to worry about token values.

Problems with Electronic Cash

All electronic cash systems have the same problems: there is a far more serious danger of counterfeiting and laundering, and some of the goals (i.e. anonymity) make these dangers far easier to achieve. Counterfeit can take place in the form of token forgery (valid looking tokens that do not correspond to a valid bank withdrawal) or the multiple spending of tokens. Systems rely on the authentication security goal (i.e. user identification and message integrity) to protect against forgery. Designers of electronic payment systems usually choose one of two ways to deal with multiple spending: they try to prevent multiple spending before it occurs, or they wait until multiple spending occurs (having systems in place to detect it) and then punish the culprits. Banks can try to prevent multiple spending by keeping a database of spent electronic tokens or coins which is checked before completing a payment. This method is useful when the transaction is done *on-line* (i.e. the merchant verifies the validity of the payment before delivering merchandise). In an *off-line* setting (i.e. the merchant deposits the payment after merchandise has been delivered) the best one can do is to detect multiple spending and identify

the guilty user so they can be punished. Even systems with prevention or detection could still incur some serious losses, for example if a user forges a token of large value, spends it, and then disappears. Some systems have been designed with some extra properties in order to limit the losses in such situations. These properties include: setting an upper limit on the value of each payment, using tamper-resistant cards that remove or disable the coin once it has been spent (assuming that generally users cannot modify the card), and having cryptographic security in place in case the tamper-protection fails. These extra features help to limit the losses that a vendor/merchant, or the Bank, may incur even when there is counterfeit protection in place.

Levels of Electronic Cash Systems

There are several different levels for an e-cash payment system. The simplest level, or form, is a system with no anonymity or untraceability that still achieves all the security goals (both on-line and off-line) through digital signatures. The next level up is a system that does not achieve anonymity, but does achieve untraceability through blind signatures, and all the security goals again through digital signatures. The word "blinding" is used to describe a user signing a coin or token with some random quantity (known only to them) before passing the token to the Bank for signing. Thus, the Bank signs something that appears random, and thus may also include the value of the token it is signing since it cannot see what it is signing. The payer can then remove the random factor and pay the payee with the token which has a valid signature from the Bank. The Bank can see that the token has a valid signature, but does not know which user paid with this token, thus achieving untraceability. The most preferable system is one which achieves the above security and untraceability features along with user anonymity. It is important to note that in order for the payer to receive a signed receipt from the payee, there is no way for both the payer and the payee to be anonymous, so we settle for user anonymity.

The problem with user anonymity is that there is no way to identify a user who is double spending tokens. In order to fix this problem, we make an adjustment so that a system achieves anonymity for honest users, but as soon as a user tries to cheat the system, their identity is revealed so the Bank can identify the culprit. This property is achieved through identifying information that is shared with the payee during the payment step. This identifying information of the payer is split into pieces such that with one piece, the identity of the payer is hidden, but their identity is revealed with two pieces. If a payer double spends a token, the Bank will receive two pieces of identifying information from the same coin, enabling the payer to be identified and

punished. There are two different ways of setting up this identifying information: cut and choose, and zero knowledge proofs.

Cut and choose is a method in which the sender/payer blinds a message of K pairs of numbers (where K is large enough in practice that an event with probability 2^{-K} will never happen in practice [2]). These pairs are created before the sender initiates contact with the merchant and the pairs have the property that the merchant can identify the sender if they possess both halves of the pair (where one half of a pair gives the merchant no identifying information). Now, when the payer gives tokens to a merchant as payment, the payee responds with a challenge question of K random bits. For each one of the bits, the payer sends the appropriate half of the K pairs they have determined in advance. These pieces are sent to the Bank with the payment. If this token is re-spent, another merchant will ask a random challenge to which the payer must send the appropriate halves of the K pairs. With a very high probability, the challenges from the two different vendors will not be the same, thus the second vendor will have obtained enough information for the Bank to reconstruct one complete pair, thus identifying the sender. Clearly this method is not very practical as the sender must construct K pairs of identifying information for *each* coin/token they create, and each token/coin that is sent to the Bank must be accompanied by K numbers from the sender.

The second method to set up the identifying information is through zero knowledge proofs which is where the sender will prove knowledge of some quantity without revealing the quantity to the receiver. The sender creates a key pair where the secret key reveals her identity. The payer sends the public key of this pair and the token to the payee for payment. The sender then proves that she possesses the secret key without revealing it to the vendor, and again if she responds to two distinct challenges (when using the same coin), the Bank can put together her identifying information to reveal the culprit. The following protocol from [2] include all the security features, untraceability, and user anonymity properties discussed thus far:

Withdrawal:
Alice (payer) creates an electronic coin, including identifying information.
Alice blinds the coin.
Alice sends the blinded coin to the Bank with a withdrawal request.
Bank verifies that the identifying information is present.
Bank digitally signs the blinded coin.
Bank sends the signed blinded coin to Alice and debits her account
Alice unblinds the signed coin.

Payment:
Alice gives Bob (payee) the coin.
Bob verifies the Bank's digital signature (thus Bob will either be paid, or learn the identity of the cheater and can refuse to deliver further merchandise to this user).
Bob sends Alice a challenge.
Alice sends Bob a response (revealing one piece of identifying information).
Bob verifies the response.
Bob gives Alice the merchandise.

Deposit:
Bob sends the coin, challenge, and response to the Bank.
Bank verifies the Bank's digital signature.
Bank verifies that the coin has not already been spent.
Band enters coin, challenge, and response in the spent-coin database.
Bank credits Bob's account.

Now that we have reviewed the properties of electronic payment schemes, we will focus our attention on payments of very little value, as small as fractions of a penny, which require micropayment systems.

3. Micropayment Schemes

A micropayment scheme, like other payment schemes, consists of protocols relating a user (or buyer), a merchant (or vendor), and a bank. The user wishes to purchase goods from a vendor for a small price, and wants to pay for only the information he receives (or downloads) without the use of tokens or coins. The micropayments in this paper refer to transactions such as paying to visit a website, or paying for each minute of streamed music or video (allowing the vendor to charge for services in small increments). The merchant wants to ensure that the user pays for the information he receives, while the user wants to ensure that he will not be overcharged. The bank assists in the transfer of funds from the user to the vendor as well as detecting and stopping fraudulent activity. We want to make secure, computationally efficient payments of small amounts from the user to the vendor (since credit card transactions can cost between 20-40 cents per transaction plus a variable fee of 2-20%). Thus, to be efficient, we need to minimize the number of public key operations, which is often accomplished by using hash functions which are approximately 100 times faster than RSA signature verification and approximately 10,000 times faster than RSA signature generation [2]. Micropayment schemes also attempt to minimize fraudulent activity, or make fraud unprofitable for any participant of the

system (user, vendor, or bank), and reduce the number of communications between the user and vendor [2] aggregating many small payments into one macropayment. Aggregation exists in two major forms: session and universal. *Session aggregation* (as we will see with Payword) occurs when a vendor collects many small payments from *one* user and submits the total as a macropayment. *Universal aggregation* (as we will see with MR3) occurs when each vendor processes payments directly (i.e. checks that the payment is valid and is "payable") and then submits the macropayment accordingly. We will discuss later how a transaction is selected for payment and how the amount of the macropayment is determined. We also note again that very few payment schemes achieve user anonymity, but all payment schemes are designed to be as simple as possible for the user and the merchant.

There are several different schemes and techniques that have been attempted in the past, and we now briefly discuss each type (as they are outlined in [2]): credit card schemes, electronic cash/cheque schemes, hardware-based schemes, subscription schemes, coupon-based schemes, and probabilistic checking and polling schemes.

Credit card schemes require all parties to be online and the bank is involved in every transaction, verifying that the user's account is in good standing and issuing a validation number to the vendor for each verified transaction. As previously mentioned, the cost per transaction is very high (20-40 cents plus additional variable costs) and the bank must always be available in order for vendors to conduct business. The principal reason for fraud with this system is impersonation by means of stolen cards or information.

SET is a scheme designed by the major credit card companies using digital signatures to authenticate all three parties. There is no impersonation with this scheme since it is practically impossible to forge a digital signature. The bank must still be present and online for all transactions, and all parties must generate and verify signatures (costly computations), making this scheme impractical for internet transactions of small amounts. *NetBill* is also an impractical online protocol that requires 8 messages per transaction, as well as an online communication with an intermediary *NetBill* server for each transaction. The major advantages of this scheme are that it provides atomicity and anonymity, although it is still unrealistic for micropayments.

The *Electronic Cash* scheme is a system in which a user gives a certain amount of money to the bank in exchange for digital cash (authenticated by the bank with a unique serial number). The biggest problem with electronic cash is the possibility of double-spending, which has been

combated in several ways, including: online verification by banks to ensure that the coins have not been previously spent, policies in which the bank only pays for each coin once, and implementations in which the user's identification is incorporated into the coin (conflicting with the goal of anonymity since the cheater can be found and prosecuted). Systems like *DigiCash* or *NetCash* prevent this double-spending by always being online.

NetCheque is another online scheme where the user issues a cheque to the vendor using public-key cryptography (i.e. the cheque is digitally signed by the user). This system requires users to register with the bank and the bank then clears each user's cheque by verifying the cheque for correctness and the account for availability of funds. Fraud once again is an issue when this verification is not done online.

Hardware-based schemes use tamper-resistant smart cards containing a private key that cannot be extracted from the card. Each vendor is given a card from a particular bank, and each user purchases tokens from the bank to be given to the vendor in exchange for goods. When a payment is given to the vendor, in the form a token purchased from the bank, the vendor's card verifies the validity of the token and the money is then requested from the bank. All vendors can verify tokens received by users through the private-key authentication tags without actually knowing the private key of the bank. A major problem occurs if a key is compromised because the security of the entire system is destroyed. *Electronic Wallets* are another version of a hardware-based system in which the user carries the card instead of the vendor. Either the card contains a counter indicating the actual value of the card, or the card itself prevents the double-spending of digitally signed coins. *Micromint* is a system introduced by Rivest and Shamir in which forging is hard and duplication is no longer an issue since the bank only pays for each coin once (although this scheme requires the storage of all coins).

Subscription schemes operate by selling subscriptions to access content from a vendor for a particular period of time. This scheme is not suitable for infrequently used vendors and most users limit the amount of subscriptions they purchase and/or are unlikely to pay high subscription fees.

Coupon-based schemes are private key solutions where the bank sells vendor-specific coins to users which are authorized using the vendor's private key. The first (unsuccessful) micropayment system, *Millicent*, was a coupon-based scheme, as is *Payword* (which will be discussed further in Section 3). These systems chain a certain number of values together such

that given any value in the chain, it is almost impossible to determine the next value, but it is always easy to verify that the chain leads back to the original value. For each payment, a user simply issues the next value in his payword chain to a specific vendor. Double-spending is again an issue that can be resolved by performing online verification, or by black-listing users who double-spend (of course, black-listing creates the problem of updating and checking lists).

Probabilistic checking and payment schemes are two more valid options for micropayments. Probabilistic checking schemes work by depositing the payments from the user at times given by a probabilistic function. This limits both the amount of time that the vendor is online and the amount of double-spending that can be done by any given user. If a user is found to be double-spending, they are black-listed and all vendors are informed (where we again have the problem of updating and checking lists). Probabilistic payment schemes require users to manage some risk. The vendors in this scheme receive payments from different users, and select only a certain percentage of these payments for deposit. The selection of payments for deposit is based on probabilistic means, and the amount of the payment is directly related to the selection rate of the payments. For example, if a vendor selects only 1/1000 payments for deposit, then the user of a selected payment will pay 1000 times more than the cost of one payment. An example of a probabilistic protocol is the *Lottery* scheme which will be discussed further in Section 4. The main idea behind the *Lottery* scheme is that the bank issues a book of lottery tickets by choosing a random x, computing $y = f(f(f(\ldots f(x)\ldots)))$, where f is a one-way hash function, and authenticating y. Each user pays with the next preimage and once the book of tickets is finished, the bank announces the winning ticket and the holder of the ticket must pay the agreed-upon price. The major disadvantage of this scheme is that the bank must check the tickets at the end of the lottery to determine which user must pay.

Another probabilistic payment scheme is a coin-flipping protocol in which a vendor and a user initiate some coin-flipping event to decide if the user pays. This scheme prevents a user from denying that the coin-flip happened (non-repudiation) by using digital signatures, thus requiring the user to pay the price which was agreed upon before the coin-flip.

We will now describe two of these schemes in detail: Rivest and Shamir's *Payword* and Rivest's *Lottery* scheme. From now on, we denote the user, vendor/merchant, and bank by *U, V,* and *B* (respectively). We denote the public keys of *U, V*, and *B* by PK_U, PK_V, PK_B and private keys by SK_U, SK_V, SK_B [9]. A message *M* with digital signature produced by *SK* is denoted

$\{M\}_{SK}$ [9]. We also let h be a strong cryptographic hash function (like MD5 or SHA-1) which is one-way and collision resistant.

4. Payword

Payword is a coupon-based scheme in which the user buys vendor-specific coins from the bank. This scheme refers to the vendor-specific coins issued (and digitally signed) by the bank as *paywords*. The bank also issues a *payword certificate* to each vendor which contains the bank's name, the expiration date, and the user's name, IP-address, and public key [9]. These certificates are renewed by the bank on a regular basis if the user's account is in good standing (i.e. they have not been involved in any fraudulent activity). With this certificate, the user is permitted to create his own *payword chain* where values in the chain are released to the vendor as payment for goods (and the values are all linked together by some hash function). The certificate also assures the vendor that they may accept paywords from a particular user. We assume in this paper that each payment has a value of 1 cent (although this could be modified in any implementation of this scheme).

For the first request from a vendor in a particular day, the user computes and signs a *commitment* which is a new chain of values (or paywords) that links the vendor to that user. The user randomly selects the last payword in the chain, w_n, and then computes the rest of the chain such that $w_i = h(w_{i+1})$ for w_{n-1} down to w_0. The user provides the vendor with a commitment which includes the first value, w_0, and his certificate. Note that the other values for w will be released one at a time (from w_1 to w_n) to the vendor as payments for goods. Since the hash function h is a one-way hash function, given any payword, it is computationally infeasible for a vendor to figure out the next payword in the chain. In contrast, each payment can be verified quite easily by taking the payment pair (w_i, i) and verifying that $h(w_i) = w_{i-1}$ (since the vendor already has w_{i-1}). Notice that this payment scheme minimizes the number of computations required since the user computes nothing, and each payment requires the vendor to compute only one hash value (along with the initial signature verification). At the end of any particular day, or when the vendor feels that the amount that a particular user has spent is sizeable enough, the vendor would send to the bank the commitment and the last payword, say w_s, received from each user. The bank can use the hash function to verify that the user did in fact request s different items (or goods totalling s cents) from the vendor, and in turn debits the user's account by s cents

and credits the vendor's account with the same amount (although bank fees may apply so the actual debit/credit may be more/less than *s* cents). The communication between the bank and the vendor is limited to once per day versus the traditional once per transaction. The vendor also has no need to contact the bank after the initial contact by the user. We now describe in detail the relationships that exist between the user, vendor, and bank, and the information which is exchanged as a result of each relationship.

The user initially contacts the bank asking for a payword certificate. *U* must provide (over a secure channel) his credit card number, his public key PK_U, and his delivery address A_U (i.e. IP-address for website purchases) [9]. The bank charges this credit card with the total number of paywords that the user gives out. The certificate issued by the bank has a specific expiration date, *E*, and will be replaced by *B* only if *U*'s bills have been paid and *U* has not participated in fraudulent activity. The certificate only authorizes purchases to be delivered to A_U, and could also contain other information, I_U. The format of the certificate is:

$$C_u = \{B, U, A_U, PK_U, E, I_U\}_{SK_B} \text{ [9].}$$

As this certificate has the user's name and public key, this scheme does not provide user anonymity, but some privacy is included since there is no record of goods purchased by a particular user. If *U* loses his secret key, he must report at once to *B* in order to limit his liability (as in the case of credit cards) and to add his certificate to the "hot-list". Alternatively, the hash of the root, w_0', could be added to each certificate and banks could provide a daily value w_j' (on day *j-1* of the month) to each user if their account is in good standing (i.e. the bank has their own chain of hash values that it releases sequentially to each user). The vendors would request w_j' on a daily basis and only provide services to those users with correct w_j' values.

Of course, there is also a relationship that exists between users and vendors. As payword chains are vendor specific, the user decides the length of the chain depending on the amount he intends to purchase from a vendor. The commitment for a chain, *M*, is sent to the vendor, and has the form: $M = \{V, C_U, w_0, D, I_M\}_{SK_U}$, where *D* is the current date and I_M is any additional information [9]. The bank is authorized (by this commitment) to pay *V* for all paywords received from *U* within a certain time period. Since payword chains (and commitments) are user and vendor specific, the user may have a significant computational burden if he quickly switches from vendor to vendor. After receiving a commitment, *V* verifies *U*'s signature on *M*, *B*'s signature on *C*, and the expiration date *D* [9]. The user cannot cheat the vendor by replaying

commitments as long as *V* keeps all verified commitments until they expire (it is also a good idea for *U* to keep all unexpired commitments sent to *V*).

As previously mentioned, a payment from *U* is the pair (w_i, i) given to *V*. Notice that this payword is unsigned because it is self-authenticating by using the commitment. The user must spend these paywords in sequential order to ensure that *V* can verify each payment by applying *h*. This ordering of payments also permits *V* to simply store the last payment received (with the largest index). Thus, if the vendor has (w_s, s), he should be paid *s* cents by the bank, which can be verified by determining how many applications of *h* must be applied to w_s to map it onto w_0. In order to pay more than 1 cent per transaction, a user can simply release more than one payword for a particular transaction or skip a certain number of paywords. For example, in order to send a payment of 5 cents, the user sends w_8 to the vendor if the last payword he sent was w_3. The vendor can confirm that the payment is 5 cents by applying five applications of *h* to w_8 and verifying that it equals w_3. The vendor could cheat *U* by not sending the merchandise purchased, or sending the wrong merchandise, but a vendor's reputation is worth far more than the merchandise he sells, and vendors who do not deliver will be avoided. If *V* specifies payment *after* delivery, *U* could then cheat the system, resulting in their name being added to the "black-list".

The last relationship to examine is that between the bank and the vendor, who do not necessarily have a prior existing relationship. The vendor must obtain an authentic copy of the bank's public key in order to authenticate certificates signed by *B* and issued to *U*. *B* and *V* must also set up a method of payment for the paywords received from *U*. *V* must send the commitment and the last payword received from each user he does business with in a particular day. The bank must verify (possibly offline) each commitment using the user's signature and the expiration date, and verify each payment which requires *s* applications of the *h* on w_s (which should map w_s onto w_0). *B* has a reasonable computational burden since hash function computations are inexpensive and signature verification computations (especially with RSA) are only moderately expensive.

We finally analyze Payword's efficiency and storage requirements with respect to each participant. The user must verify certificates, sign each commitment he creates, and perform one hash function operation per payword chain value. He needs to store his secret key, SK_U, his active commitments, the corresponding payword chains, and his current position in each chain

[9]. The vendor must verify all certificates and commitments received from users, and perform one hash function application for every payword received (or skipped over if payments are greater than 1 cent). He needs to store all commitments received from users and the last payment received per day from each commitment [9]. The bank needs to sign each certificate issued to a user, and perform one hash function application per payment released from a user to a vendor. He needs to store copies of the user certificates that he has issued, and maintain accounts for users and vendors [9].

This scheme could be modified in several ways. Each user could have many different payword chains, each chain having a different transaction value. Paywords could be sold to users on a debit basis, where the bank would refund the user for paywords in each chain that they did not use. The bank could also limit the amount of money that a particular user can spend to buy merchandise from a particular vendor (this would be specified in *U*'s certificate).

The benefit of using this system is that we do not have to send all the payment information to the Bank with each transaction since the chain of values (which is computed before the transactions occur) is used for multiple transactions with a particular vendor. As previously discussed, we also satisfy the divisibility property because the chain of values ensures that the user's credit card is charged exactly the amount they spent. This system also meets all the security properties and the chain of values is recognizable as *legal tender* through verification.

As secure as this payment scheme seems, there are still some major problems that do not make this scheme practical in the world of micropayments. The merchant cannot aggregate payments of multiple users [2]. Each user creates a *unique* chain of values with each merchant with whom they do business. Therefore, if a user contacts a merchant to view only one website, the merchant would either have to process this transaction and lose money, or let the user have it for free (which would soon become a problem if users discovered that they could see the first 20 websites for free). This scheme is also not transferable, not untraceable, and not anonymous (although untraceability can be added as previously mentioned). The new payment schemes proposed by Rivest and Micali in Section 5 will fix these problems.

5. Lottery Scheme

Rivest's "Lottery Scheme" is one of the few payment schemes in which the bank only sees a percentage of the total payments sent for deposit. This scheme works in the exact opposite way of an ordinary lottery, since the user/issuer of the winning ticket must *pay* the vendor, and the vendor/purchaser pays for the tickets with information. The selection rate, s, is between 0 and 1, and determines the probability that a ticket will be a winner. A non-selected (non winning) ticket is worthless, while a winning ticket is worth $1/s$ times more than the original amount of the merchandise [5]. A "ticket" contains the following information: the name of the issuer, the name of the buyer, the name of the recipient (vendor), a ticket number (usually randomly selected, short, and determined by the user), a winning number indicator (indicating how the winning number is determined), a ticket face value (indicating how much the buyer is required to pay if he is the winner, such as $1/s$ times the original value), the name of the payer (who makes the payment), and a ticket credential (ensuring the vendor that the payer will pay) [7]. Each ticket is signed by the issuer, which is usually the user. The buyer (user) wishes to purchase some merchandise or information from a vendor, and the recipient (vendor) exchanges the user's ticket for the goods. The winning number indicator can either be internal (such as the last three digits of the ticket number having a specific hash value) or external (some source or authority who announces winning numbers). When using internal indicators, it is easiest for the recipient (vendor) to randomly create a unique number, w, which will be used to indicate which tickets are winners, and issue $h(w)$ to the user before he issues the ticket. Regardless of what type of indicator is used, the issuer cannot cheat the system because he should have no idea if his ticket is a winner at the time he issues the ticket (i.e. knowing the hash value of w should give no indication of the last three digits of w). The face value of the ticket is large enough so that the bank's processing costs are small in comparison to the amount the user pays (i.e. $10). The expected value of the ticket is the price of a transaction if every user paid, which is calculated by multiplying the percentage of winning tickets together with the face value. The payer (bank or credit card company) co-ordinates the payment of the face value from the user to the vendor. The ticket credential could be a signed statement from the bank (payer) that they will pay for the user's (issuer's) purchases for a given time period (i.e. a month). This means that the user has enough money/credit to pay all tickets issued within a certain time period. This scheme drastically reduces the bank's processing costs, while requiring only moderate computations from

the vendor and user for each transaction. There is always a risk that too many, or too few, tickets will be winning tickets, but this probability is very small. For example, if 1 in every 100 tickets wins, the user should expect to pay, on average, $1 for every 100 websites visited (if they each cost 1 cent). Users can easily track their purchases if vendors notify them immediately about winning tickets (which can be done when internal indicators are used).

Let us now examine the basic Lottery scheme in detail, outlining the relationships that exist between the user (issuer/buyer), vendor (recipient), and bank (payer). The standard version of this scheme has the user as both the issuer of the ticket and the buyer (i.e. has to pay if his ticket wins). The user gets a signed ticket credential from his bank guaranteeing that his account is in good standing. The vendor creates an internal indicator by selecting a random number, and sending the hash of this number to the user to include in his ticket. There is usually no limit to the number of tickets that a user can create, or the number of vendors with whom a particular user can do business. In the chance that a user issues too many winning tickets and does not have enough credit to pay for all of them, some vendors will simply not get paid. The bank (payer) will not re-certify this user in the next time period. Vendors can adjust their prices (over time) to cover losses for unpayment, although one unpaid ticket is not a debilitating loss to the vendor when dealing with micropayments. If using an internal indicator, the vendor knows immediately if a user has a winning ticket, and can instantly deposit this ticket to verify that the user has sufficient funds. If the user does not have enough cash/credit, the vendor can deny him services for the remainder of the time period. The user must also manage some risk, realizing that he may generate more than his share of winning tickets. The probability that a user will be billed for much more than he expects is very low, but the amount owed by any user will vary. With internal indicators, a user can at least see his winning tickets immediately to help track his spending.

Another version of this protocol uses a "Payword" chain (discussed in Section 3) to create electronic lottery tickets. The random number w given to the user from the vendor becomes the root node, w_0, of one payword chain. The user also has his own payword chain with values x_i (with x_0 being the root) indicating different tickets. A user commits to a vendor by sending the first value in the ticket chain, x_0. Each payment is made by releasing the next value in the chain, which the vendor then verifies satisfies $h(x_{i-1}) = x_i$. This ticket is payable if and only if $x_i \ mod \ 1000 = w_i \ mod \ 1000$, where the selection rate here is 1/1000. The user can verify that x_i is

a winning ticket by applying h to the payword chain of w values (given by the vendor), and can never determine if his ticket is a winner before sending it.

The bank could also be the issuer of tickets, or authorize a batch of tickets to be created by the user/buyer [7]. The user would still release each ticket to the vendor for payment, with the additional risk that a user could double-spend a ticket and give it to more than one vendor. Note that the buyer must still sign each ticket over to the vendor [7].

The standard scheme can be modified to use external indicators as well, with the additional risk of collusion between the bank and the user if the bank creates the winning ticket numbers. If the user knows the winning numbers before he creates his own tickets, he can make sure he does not create any winning tickets, thus defrauding the vendor. A resolution would be to create daily winning numbers by hashing a number of independent sources together so that the bank cannot predict their values. Vendors must also store all the tickets they receive in a given day until the winning number is released, which could be a huge storage burden on the vendor, not to mention that the user no longer receives immediate notification if he holds a winning ticket.

We finally analyze the "Lottery" scheme's efficiency with respect to all participants. Since the user must issue an electronic ticket for each micropayment, this computation must be fairly rapid, which can be done using techniques similar to those in Payword which use only a small number of public-key operations, along with one signed message for each vendor [7].

The vendor's computations are also comparable to other micropayment schemes, with his storage requirement minimized since he keeps only winning tickets. This also improves user privacy since only a portion of the tickets he issues are actually kept. Some of the stored tickets may actually be worthless if a user over-spends his credit, but vendor's losses will be minimized if they deal only with reputable banks [7].

The bank has only two jobs: providing monthly credentials to customers (with accounts in good standing), and paying for winning tickets. The bank does not need to perform withdrawal protocols because they charge the user's credit card for micropayments associated with winning tickets. The electronic tickets are very efficient for the bank due to the probabilistic aggregation of many small micropayments into a larger payment from a winning ticket.

It is important to note that true anonymity is quite costly in this scheme (even more so than any other scheme) as it requires an intermediary who would interact with the user and the vendor to issue tickets and report winning tickets. Anonymity could also be achieved by using pseudonyms known only by the user and the bank, and using an intermediary to conceal the network address from the vendor [7]. The paper cash properties included in this scheme are again divisibility (indirectly) and recognizable as legal tender. As previously mentioned, the lottery scheme also improves user privacy as the bank/credit card companies can only see a small fraction of their purchases.

There are still some problems associated with this payment scheme such as the interaction required between the vendor and the user to select a winning ticket, and the risk that must be managed by the user since he could pay more than he should [5]. This last problem is more of a psychological problem as the probabilities work out such that a user should pay, on average, the correct amount. Further problems were previously described when discussing external indicators (i.e. large storage requirement and users conspiring with the bank). These problems make this scheme impractical for real world use and solutions will be discussed in Section 5. Again, the standard scheme is not anonymous (but could be made anonymous at a cost), transferable, or untraceable. Most users would choose Payword over Lottery due to the risk that they must manage with Lottery (although there is a trade-off concerning their privacy). Most merchants would prefer the Lottery scheme over Payword since they would receive *1/s* times the transaction value for *each* payable cheque, thus the unpaying users have been taken into account (unlike Payword when a user only buys *one* item from a merchant).

6. New Micropayment Schemes

Recent micropayment systems have arisen out of the need to address the problems inherent in earlier micropayment schemes. These new schemes fix the problems with the "Payword" and "Lottery" schemes, provide a more user friendly and simple interface, provide more efficiency than the traditional schemes, and use the probabilistic deposit protocol in order to reduce bank processing costs. We call these new schemes MR1, MR2, and MR3 as introduced in [5]. The MR1 scheme improves upon the "Lottery" scheme by making it non-interactive, while still informing the user immediately if he issued a winning ticket. The MR2 scheme modifies the charging protocol to ensure that a user will not be charged more than he

ought. The MR3 scheme gives the bank greater control over the deposit process but gives less immediate results to the vendor concerning which cheques are payable. Each micropayment, in all three schemes, requires a digital signature computation which is quickly becoming a non-issue in computational costs due to more powerful processors, and the more secure and efficient signature schemes that are currently available. Online/offline digital signature schemes may be a good choice for micropayment schemes, where the offline step is more computationally demanding and performed before the message to be signed is chosen or available, and the lightweight online step is done when the message is available [2].

MR1

This scheme requires no interaction between user and vendor, while the vendor still learns immediately if a ticket is a winner (or if the cheque is payable). Thus, a ticket/cheque, *C*, issued by a user to a vendor will be payable if a certain property holds between *C* and a unique quantity dependent on *C* that is easily computable by the vendor, but unpredictable to the user [5]. This means that any ticket/cheque sent to the vendor is already "pre-selected" for payment if this property holds. This property is different from the "Lottery" scheme where payability depends on the random choices made by the user and vendor rather than on properties of the ticket/cheque, although the determination of payability still occurs *after* the protocol has finished. This protocol has the following three properties: the fraction of cheques marked "payable" is approximately *s* (i.e. the probability that the property is satisfied is approximately equal to the selection rate), the user cannot read if the cheques he sends are payable or not, the vendor can read immediately if the cheques are payable, and can make this mark visible to others [5]. We now describe the standard version of MR1, followed by some variants of the scheme.

We let *T* denote (the encoding of) a transaction, including all useful properties such as the user, vendor, bank, merchandise, transaction time, transaction value, etc [5]. Again, we assume that each transaction has a fixed value of 1 cent (which can be modified if necessary) and the selection rate (proportion of transactions deposited) is *s*. We describe a new function, *F(.)* which takes arbitrarily long bit strings as input and outputs a number between 0 and 1 (inclusive). For example, *F* could pre-pend a zero to the beginning of the string and interpret the string as a decimal number (i.e. input 001, output 0.001). We denote *U*'s signature on *M* to be $\{M\}_{SK_U}$ as in

Section 3 (although some papers prefer to use $SIG_U(M)$ instead), and we assume that each message has an application of *h* prior to signing.

The set-up for this scheme is easy since each participant only needs to establish a public and private key. We note that the vendor's signature scheme must be deterministic [5] (such as the hash-then-sign version of RSA) to prevent cheating, and this scheme does not require a separate set-up for each user/vendor pair.

The payment between the user and vendor happens in the usual way with *U* sending the cheque $C = \{T\}_{SK_U}$ to the vendor for transaction *T*. *C*, or a portion of *C*, is authenticated and if *U* wants an anonymous purchase, *C* can be authenticated and sent by a third party. The digital signature scheme may be deterministic, randomized, identity-based, off-line, on-line, etc. The merchant must first verify *U*'s identity using measures such as *U*'s certificate, the certificate expiration date, checking if the certificate has been revoked, checking if *U* is authorized to write *C*, etc. The cheque is payable if $F(\{C\}_{SK_V}) < s$, which means that *V* then sends the bank both *C* and $\{C\}_{SK_V}$ for deposit. *V* could also notify the user that his cheque was payable, without forfeiting the non-interactivity property since the vendor does not *need* to send anything back to the user. Also note that the value of $\{C\}_{SK_V}$ is unpredictable and virtually random to *U*, thus he cannot predict the value of *F* (i.e. if a cheque will be payable or not). Thus, $F(\{C\}_{SK_V})$ will be a random number between 0 and 1, and will be less than *s* for a fraction, *s*, of the cheques sent [5]. If the signatures on *T* and *C* are correct, and *C* has not been previously deposited, then the bank will credit *V*'s account and debit *U*'s account with *1/s* cents. The bank can justify this action to *U* by providing the cheque, *C*, that he sent to *V*, along with *V*'s signature on *C*. Note that the bank is only called into action for a fraction (1/*s*) of the cheques sent to *V*, and each payment is now a macropayment. Most importantly, no two parties can collude against the third party. If *U* had *B*'s help, he still cannot write a cheque that has a smaller chance of being deposited (since both *B* and *U* cannot predict *V*'s signature on *U*'s cheque). The vendor and bank cannot collude against the user either. Since *V*'s signature scheme must be deterministic, there is only one correct value of the signature for each cheque sent by *U*, and when *V* chooses his public key, he has no idea what the cheques will look like (*U* has an unpredictable signature) so he cannot force more than 1/*s* of the cheques to be payable. Finally, *U* and *V* cannot collude against the bank since *B* debits and credits the accounts with equal values. There is always the risk that *U* will not have enough

money to pay for the transaction, but this is an unavoidable risk in any payment scheme, so it is independent of these micropayment schemes and should be handled as usual [5]. It is also important to note that there could be four parties involved in this scheme instead of three. For example, we could have a music downloading system where the parties are the downloading user, the provider, the user's Bank, and the music distributor.

Both theoretical and practical variants of MR1 exist. In the analysis of MR1, it is crucial that $F(\{C\}_{SK_V})$ is a random number that is unpredictable to the user. This number will depend on the definition of F and the security of the signature scheme (i.e. signature schemes or hash functions could be modelled as random oracles). When using a secure signature scheme such as RSA, it has been proven that with a randomly chosen public key of L bits, the last $c*log(L)$ bits of the signature on a random message are computationally indistinguishable from a random $c*log(L)$ bit-string, where c is a constant greater than 1 [5]. For example, if a vendor uses public and private keys of 1024 bits, and chooses $c=2$, then the last 20 bits of his signature on *any* cheque C will be indistinguishable from any random 20 bit string. Note that using as few as 20 bits enables this scheme to be implemented with selection rate as low as 2^{-20} which allows micropayments of 1 cent to produce macropayments of $10,000 [5]. The vendor could also use "verifiable random numbers" as described by Micali, Rabin, and Vadhan [4].

Practical variants of this scheme, which still provide the non-interactivity of MR1 are as follows. The vendor could offer subscriptions to his service (which gives unlimited access for a certain amount of time) or purchasing items "à la carte". If a user chooses to only purchase items "à la carte" and ends up paying more than the value of a subscription, he then gets a subscription for free. The bank could modify the protocol to only deposit cheques with a date within one day of the current date, thus preventing vendors from charging users when they are no longer expecting to be charged (preventing surprise charges on the user's account). Therefore, the vendor should always verify the time accuracy of the cheques sent by U before sending the requested merchandise. For example, B can refuse V's payable cheques unless they have the correct time information. Each transaction would then contain information about the time the transaction occurred, t, and the property that determines payability would also be with respect to this time quantity. Thus, B can refuse a cheque if $|t'-t|>$ some predetermined quantity, where t' is the time of deposit. This protocol could also be modified such that the cheque payability condition $F(\{C\}_{SK_V}) < s$ could be replaced with $F(\{G(C)\}_{SK_V}) < s$ where functions F and G

could vary from transaction to transaction. This means that *V* could sign a quantity dependent upon *C*, say *G(C)*, rather than on *C* itself [5], where *G(C)* could also be a function of *T* alone. The cheque payability condition could also refer to some property dependent upon *C* rather than on *s*. For example, the vendor could say that a cheque is payable if the last ten digits of *C* equal the last ten digits of the vendor's signature on *C*, or the vendor's signature on *G(C)*. Another variant, which minimizes the number of signatures that a vendor must compute, determines cheque payability using $F(\{G(V_i)\}_{SK_V})$, where $\{V_i\}$ is a sequence of values associated with a sequence of times, rather than using $F(\{G(C)\}_{SK_V})$ which requires a separate signature computation for each *C* sent by *U* [5]. For example, V_i could be a daily value specifying the date and a cheque *C* related to transaction *T* on day *i* is payable if $F(\{G(V_i)\}_{SK_V}) < s$, or if some other property holds between *C* and a quantity computable from V_i [5]. The vendor could evaluate $F(\{G(V_i)\}_{SK_V})$ at the beginning of each day in order to determine immediately upon receipt of *C* if it is payable, discarding it if not. It is important that the vendor hide all discarded and deposited cheques for a given time period, or the user may make predictions about which cheques are payable. It is also crucial that the vendor keep all payable cheques until the end of a given time period, depositing everything at the end of the period, or else the bank could collude against the vendor. A variant on this V_i approach is to compute a chain of values as in Payword with the root stored in a public directory. The vendor could then simply use the *i*th value in the chain on day/time *i* instead of $F(\{G(V_i)\}_{SK_V})$. This approach is also scalable for payments of various sizes [5].

MR2

This scheme deals with the psychological problem that users have when dealing with risk, thus attempting to make the micropayment scheme more widely accepted (and thus used) by users. The scheme presents a selective-deposit protocol that solves both problems with the Lottery scheme, shifting the management of risk onto the bank, and guaranteeing that a user will never pay more than he ought. The problem of risk does not bother the bank, who is in the habit of managing huge risks and realizes that the risk of overpayment becomes less and less probable in the long run [5]. This is also an extremely simple scheme which does not try to prevent

cheating, but rather punishes all those who are caught cheating before they can create huge losses.

The set-up phase is again very easy with all parties establishing public and private keys to be used in digital signatures. Again, the vendor's signature scheme must be deterministic to prevent them from cheating.

U again sends $C = \{T\}_{SK_U}$ to *V* for transaction *T*, although now the time and a serial number are included in every cheque (where the serial numbers should be assigned sequentially, starting with 1). A cheque, *C*, is again payable if $F(\{C\}_{SK_V}) < s$, which again means that the vendor then sends *C* and $\{C\}_{SK_V}$ to the bank. In this scheme, we let *maxSN* denote the maximum serial number of a payable check sent thus far from *U* to *V*, where initially *maxSN* = 0 [5]. If *C* is a payable cheque from *U*, then *B* will deposit *1/s* cents into *V*'s account provided that the signatures on *T* and *C* are both correct. If the serial number on *C* is greater than $maxSN_U$, the bank debits *U*'s account by *SN-maxSN* cents (or *(SN-maxSN)*TV* where *TV* is the transaction value when we allow transactions worth more than 1 cent), and sets *maxSN=SN* [5]. The bank may give *C* and *V*'s signature on *C* to *U* to justify this action. The bank may fine a user or throw him out of the system if this new cheque either has the same serial number as a previously deposited cheque, if the serial number and the time are "out of order", or if the amount of the cheque is excessive, etc [5]. The bank may keep statistics on users and throw them out if their cheques cause exceptions in the system. The bank may also throw out vendors who have more frequently payable cheques.

As with the previous scheme, the set-up phase for MR2 is very simple and the payment phase is non-interactive (i.e. the vendor need not respond to the user). This scheme also guarantees fairness to an honest user and will not overcharge. At any time, an honest user has been charged at most *maxSN* cents with *maxSN* being the highest serial number on a payable cheque issued by *U*. If *U* has made *n* transactions by time *t*, then $max\ SN \le n$ which means that *U* will be charged *at most n* cents for his *n* transactions. In this scheme, it is better for a user to be ignorant of which serial numbers have been payable, thus the vendor should not inform *U* which cheques were payable as the user's charges do not depend on which cheques were payable, but only on the number of cheques he has made up to a certain point in time. If a user uses the same serial number twice, then *B* can catch (and therefore punish) *U* by finding two payable cheques issued by *U* with the same serial number or a serial number that is out of sequence with

the time (*B* can check the time of the last payable cheque to the current time). A user and vendor may collude in this scheme if *V* sends, in secret, his signature on *C* before *U* issues the cheque. With some trial and error, *U* can write only payable cheques and share the illegal profits with the vendor [5]. The user pays 1 cent for each transaction, while *B* must pay *1/s* cents to *V* for every cheque. If *U* and *V* repeat this scam several times, they will be thrown out of the system as soon as *B* detects that there is an increase in the number of payable cheques to *V*. *B* can also cover losses from excessive payments to *V* by charging *U* to set up an account with *B*. *B* must make it difficult for a cheating user or vendor to re-enter the system (i.e. by imposing a harsh fine) so that this illegal activity has very little gain for cheaters, and the loss is easily absorbed by the bank. If an honest user unintentionally creates more payable cheques (which will happen infrequently), the bank may throw him out of the system and convince him of the losses he incurred to the bank by providing the vendor's signatures on his cheques. The bank may keep these users in the system under conditions of the MR1 scheme where the user will then be debited *1/s* cents for each payable cheque issued.

Once again, there exist variants to this standard MR2 scheme, the majority of which are the same variants to the MR1 scheme. For example, the payability of *C* may depend on the value of the cheque if different cheque values are supported. Two different ways of calculating the payability condition are as follows. The step function (for a transaction with value *v* cents) makes a cheque payable with probability *1/100* if $v \leq 100$ and with probability 1 if $v>100$. The ramp function (for a transaction with value *v* cents) makes a cheque payable with probability *v/1000* if $v \leq 1000$ and with probability 1 if $v>1000$. A new variant associated only with MR2 is the handling cheques of different values. We can either bundle a certain number of 1-cent cheques together, or just increase the serial number by more than one. For example, a bundle of *v* cheques can be seen as one *v*-cent cheque [5], or we can increase the serial number from *SN* to *SN*+*v*. The user is then debited by *SN*+*v*-*1*-*maxSN* and *maxSN* is set to *SN*+*v*-*1*.

MR3

This third scheme differs from the previous two by giving the bank more control over the deposit process, determining both probabilistically and fairly which cheques are payable to the vendor [5]. Again, the risk of excessive overpayment is handled by the bank instead of the user (which makes this scheme more accepted by users). As in the MR2 scheme, cheaters are

punished or eliminated from the system before they can cause huge losses (instead of trying to prevent cheating). The set-up is similar to the other MR schemes with each party establishing public/private key pairs. Payment is exactly like the MR2 scheme where each user includes a progressive serial number in each cheque sent to a vendor.

This scheme presents a new deposit protocol since the bank now has control over the probabilistic deposits. This scheme also uses a commitment scheme which is a protocol allowing a person to deliver a message to a second party without revealing the contents but still being "committed" to the message (i.e. the second party is assured that the sender cannot change the message). The second party can only read the committed message when given a "key" to unlock it. We let t and t' be the times of the vendor's last and current deposit [5]. The vendor groups together all cheques received between times t and t' and divides these cheques into n disjoint lists $L_1, L_2, \ldots, L_n$ where the cheques in the ith list total V_i. The vendor then sends n commitments to the bank, $C_1, C_2, \ldots, C_n$ where we compute each commitment as $C_i = h(L_i, V_i)$. B cannot compute L_i from the hash. He also includes the deposit time and puts his signature on the entire package. For example, B receives $\{t, n, V, h(L_1, V_1), h(L_2, V_2), \ldots, (L_n, V_n)\}_{SK_V}$. The bank verifies the deposit times and sends k randomly selected indices $i_1, i_2, \ldots, i_k$ to the vendor indicating the lists that he should de-commit so B can read the appropriate transactions. The bank then credits the vendor's account with T cents (where $T = sum(V_i)$) and debits user's accounts who have cheques belonging to lists $L_{i_1}, L_{i_2}, \ldots, L_{i_k}$ according to the serial numbers in their cheques. The bank may fine a vendor or throw him out of the system if there is a list L_{i_j} whose total differs from V_{i_j}, or if a cheque in L_{i_j} has the wrong time [5]. The bank may also fine or throw out a user if a new cheque has the same serial number as a previously deposited cheque, if the serial number and the time are "out of order", or if the amount of the cheque is excessive, etc [5]. The bank may again keep statistics on users and vendors, throwing them out of the system if their lists or cheques cause exceptions.

It is important to note that the bank decides the value of k and if it suspects that there is fraudulent activity occurring, it will increase the number k to compensate for the fraud (until it catches the cheaters). The bank could ask a vendor to de-commit all of his lists (i.e. $k=n$) for verification and pay the vendor only a fraction m/r of the aggregate values (where r is the number of lists selected for repayment and the user is debited m times the face value of the

cheque). We could also debit the user with respect to serial numbers as seen in MR2. Generally, we use k such that $1 < k < n$, where we need $k>1$ in order to catch two cheques with the same serial number. The vendor has the option of depositing cheques whenever he desires, usually this is done if he has acquired a certain number of cheques, or if the cheques received total a pre-specified amount.

Variants of the previous two schemes may also apply to the MR3 scheme, in addition to the variant where vendors can use Merkle trees to commit his lists, and authenticate at each node the total value of the checks "stored below it" [5]. The root of this tree could also be digitally signed by each vendor. The value of n could also variable in this scheme. The bank may choose to de-commit k of n lists and only credit the vendor's account with an amount that is dependent on k, n, and the value of the cheques contained in the k de-committed lists [5]. The bank and the vendor could agree on an upper bound that deposits cannot exceed (no matter how much the cheques total), or an upper bound for the number of cheques in a given list.

The three schemes discussed in this section do not have the same problems associated with the Payword and Lottery scheme. They include the paper cash properties of divisibility, recognizability, and again they could be modified to include untraceability. The MR3 system is still not transferable or anonymous (although recall that anonymity can be added for a cost). On the positive side, these new schemes provide a simple user interface, efficient computations, and prevention of malicious behaviour by any of the three parties. The MR3 scheme provides vendors with the flexibility of aggregating the cheques of many users into one macropayment. We will now briefly describe an application of this scheme that has been implemented in the "Peppercoin Payment System".

7. Peppercoin

The founders of Peppercoin (Rivest and Micali) claim that it is the "cheapest and most secure way to buy and sell digital content on the Web" [11]. Peppercoin is a payment system for low-priced digital goods and services such as music, videos, and games, that is based on the MR3 payment scheme. The MR3 scheme allows Peppercoin to efficiently process small payments while ensuring security and fairness [11]. This payment scheme acts as an intermediary between the user and the vendor, encrypting the vendor's content to sell and allowing users to purchase the content (giving them the key to decrypt the content).

PepperBoxes are encrypted digital contents that vendors wish to sell and clients purchase and download onto their computers. Users use a simple interface called a *PepperPanel* with which they interact to get keys to extract content from the PepperBox, guaranteeing that they only pay for the same content once. PepperBoxes are purchased using PepperCoins, and once the user chooses to "Save" the PepperBox, the content has already been purchased and the user is billed (even if the user cancels the download, they will still be billed for the content) [11]. When a user downloads content from a vendor, he creates a unique PepperCoin to pay for it, identifying the user as the purchaser and the vendor as the party to be paid. These PepperCoins are only valid at the time they are created, thus they are useless if intercepted. As in the MR3 scheme, PepperCoin reduces the need to process certain payments, thus reducing the workload and costs associated with the system [11]. The designers claim that their system is 10 times faster than other payment systems because they process only 1 in 10 transactions (which can be modified), although when using serial numbers as in the MR3 scheme, each client will pay a maximum of *maxSN* cents for his transactions up to a certain point in time.

PepperCoin has been running successfully since February 2003, adding to their repertoire further products for 99 cents in January 2004. This scheme, using the probabilistic deposit protocol of MR3, successfully aggregates micro-purchases into macropayments. Users can use a single PepperCoin across multiple vendors, and vendors can aggregate payments of multiple users, making this system universal and convenient [11]. In fact, the founders state the following theorem in their paper introducing Peppercoin [8]:

Theorem 1: If a Peppercoin implementation which charges a fee of pT for processing each transaction of value T, while another system that charges qT for processing each transaction of value T, then once the total number of macropayments (qualifying micropayments) exceeds $(5 / (q - p))^2$ the probability is 999,999 out of 1,000,000 that the merchant's net total receipt will be higher with Peppercoin than with the other system.

For example, let's consider a Peppercoin system charging one penny to process a 10 cent transaction (i.e. $p=0.1$) and the competitor offers to process the same transaction for 3 cents (i.e.$q=0.3$). According to the theorem, after only $(5/(0.1-0.3))^2=625$ macropayments, the vendor will be further ahead with the Peppercoin system. There has been no proof published for this theorem.

8. Conclusion

In order for a micropayment scheme to have a successful future, it must be computationally efficient, fair, and user friendly. This paper presented two old impractical micropayment schemes: Payword and the Lottery scheme which Micali and Rivest have fixed in their new MR3 scheme which has been successfully implemented in the *Peppercoin Payment Scheme*.

This paper has presented only a small sample of micropayment schemes, and does not claim that these are the only practical schemes. There are more schemes that seem promising, such as Odlyzko and Jarecki's scheme [1] which also modifies *Payword* by adding a different probabilistic protocol for deposit. There is also a scheme introduced by Payeras-Capellà, Ferrer-Gomila, and Huguet-Rotger [6] which includes anonymity while using a double-spending prevention algorithm. In most schemes, anonymity is traded with efficiency, but this proposed scheme claims to incorporate both. The final type of protocol worth mentioning is the spending program with reusable coupons which has floating scrip that is used from vendor to vendor. Schemes like *NetCents* (with it's many variants) operate on this premise.

Micropayments do have a future, and successful payment schemes will give businesses the opportunity to generate extra revenue by collecting pennies at a time.

9. References

1. Stanislaw Jerecki and Andrew Odlyzko. An efficient micropayment system based on probabilistic polling. MIT / AT&T, February 1997.

2. Laurier Law, Susan Abett, and Jerry Solinas. How To Make a Mint: The Cryptography of Anonymous Electronic Cash. http://jya.com/nsamint.htm

3. R.J. Lipton and R. Ostrovsky. Micro-Payments via Efficient Coin-Flipping. In *Financial Cryptography Proceedings 1998*, LNCS 1465, Springer-Verlag, pp. 1-15. Springer, 1998.

4. Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *40th Annual Symposium on Foundations of Computer Science*, pages 120–130, New York, October 1999. IEEE.

5. Silvio Micali and Ronald L. Rivest. Micropayments Revisited. In *Proceedings of the Cryptographer's Track at the RSA Conference 2002*, LNCS 2271, Springer-Verlag, pp. 149-163, Springer, 2002.

6. Magdalena Payeras-Capellà, Josep Lluís Ferrer-Gomila, and Ll. Huguet-Rotger. An Efficient Anonymous Scheme for Secure Micropayments. In *ICWE 2003 Proceedings*, LNCS 2722, Springer-Verlag, pp. 80-83. Springer, 2003.

7. Ronald L. Rivest. Electronic Lottery Tickets as Micropayments. In *Proceedings of Financial Cryptography 1997*, LNCS 1318, Springer-Verlag, pp. 307-314. Springer, 1997.

8. Ronald L. Rivest. Peppercoin Micropayments. Computer Science and Artificial Intelligence Laboratory Massachusetts Institute of Technology Cambridge, MA 02139 citeseer.ist.psu.edu/676873.html

9. Ronald L. Rivest and Adi Shamir. Payword and MicroMint: Two simple micropayment schemes. In *Proceedings of 1996 International Workshop on Security Protocols*, LNCS 1189, Springer-Verlag, pp. 69-87. Springer, 1997.

10. Nicko van Someren (Moderator), Andrew Odlyzko, Ron Rivest, Tim Jones, and Duncan Goldie-Scot. Does Anyone Really Need MicroPayments? In *Proceedings of Financial Cryptography 2003*, LNCS 2742, Springer-Verlag, pp. 69-72. Springer, 2003.

11. www.peppercoin.com