Microcash: Efficient Off-Line Small Payments

Chris Pavlovski and Colin Boyd

Information Security Research Centre
School of Data Communications
Queensland University of Technology
Brisbane, Australia

chripavl@au1.ibm.com, boyd@fit.qut.edu.au

Abstract. An off-line electronic cash scheme is proposed that is suitable for small payments. The approach is innovative, in that each coin may be efficiently verified by the same or different merchants during payment. The scheme relies on a batch signature technique to efficiently sign and verify individually spent coins; coins may also be deposited in batch manner. The scheme outlined differs considerably from conventional micropayments schemes by servicing a number of cash-like properties, such as off-line processing, detection of double spent coins, and ability to spend at different merchants. Additionally, the scheme eliminates a number of processing overheads that are apparent to some existing micropayment schemes.

1. Introduction

Electronic commerce schemes appear to be carving three distinct markets within the Internet environment. This includes electronic cash, electronic payment, and electronic micropayment schemes. Electronic cash schemes are equipped with tangible objects that are analogous to real world cash, by way of notes and coins of some financial denomination. Some notable schemes are described in [3, 4, 15] and exhibit a number of additional properties over and above their electronic payment counterparts including transferability, untraceability, and divisibility.

In the case of electronic payment, or macropayment, an approval or commitment is given for a financial amount. Essentially, the customer offers something which is akin to a signatory response of their commitment to a payment. Furthermore, the mechanism for extracting the payment for this commitment often forms part of the overall protocol. A salient list of such schemes includes [2, 6, 22], providing comprehensive signatory and commitment protocols, often with an associated financial collections mechanism, such as vendor specific or credit card accounts. Due to the associated costs of these schemes, both financial and computational, a floor limit to the payment exists, making certain commercial applications impractical. As such, a third breed has emerged offering a solution to the costs per transaction boundary, and addressing a ready market unreachable to macropayments.

Micropayment schemes have evolved to address two areas remaining exposed by macropayments solutions. Given the frequency and financial amount associated with some charging mechanisms, it becomes impractical for certain schemes to deal with transaction processing and revenue collections of these small amounts. Basically, the relative financial and computational cost of performing the transaction, is not worth the financial gain from the commitment. Secondly, payments must be created and verified efficiently by the customer and merchant respectively. General approaches to solve these problems include a roll-up of the smaller amounts into one large amount for an existing collections method, and to mitigate certain processing requirements to improve efficiency.

This paper proposes a new off-line payment scheme that provides an efficient cash protocol for conducting many small payments. Each coin in our scheme may be efficiently verified by independent merchants. As such, we view the main properties of our new cash scheme:

- *Off-Line* cash scheme offering efficient small payments.
- Coins are not restricted by some form of validity period, as such there is no need to perform a *coin return* protocol for unspent coins.
- No excessive precomputation activities, wastage of coins or sticks as found in other schemes.
- Coins may not be double spent.
- Coins may not be spent by a false identity claiming to be the withdrawing customer.
- Unlike micropayment schemes, each coin may be verified independently by different merchants.
- There is no requirement to check for stick or coin replay.

The following section outlines the general mechanisms currently employed by the various existing micropayment schemes. This is then followed by a review of batch cryptography how this may be applied to devise a new efficient payment scheme. Microcash is then described in detail, followed by a characterisation of the security and efficiency.

2. Micropayment Mechanisms

A number of micropayment protocols have been proposed in the literature [1, 2, 5, 9, 10, 14, 19, 17, 18, 23]. Each scheme offers a unique solution that satisfies one or more requirements common to micropayment transactions. This includes computational efficiency, client responsiveness for rapid successive payments, and cost-effective commitment conversion. Some key mechanisms employed by these systems include hash chains, probabilistic inspections, and the avoidance of costly cryptographic primitives.

2.1 Probabilistic Verification

As the processing of individual transactions may be quite expensive, one efficiency approach is to perform processing on some random basis that yields an expected result, such as lotteries or bets [18, 23]. The idea is based upon the probability that a valid payment event may occur, in a manner that when a number of transactions are performed the value of the payment averages to a correct nominal amount. For example, one valid $10 payment over 1000 payments averages to 1¢ per transaction. Here the customer forwards to the merchant some random payment, with a certain probability of being valid. Only on receipt of a valid payment does the merchant perform the normal processing. Of importance to such schemes, is confidence that the probability of capturing the correct funds, over a number of transactions, is quite high.

2.2 Redeemable Tokens

Redeemable commitment tokens, such as coins, involve some precomputed activity conducted by some minting firm, such as a bank. The coins themselves must be assembled for quick and efficient verification. Some notable scheme rely on both hash operations for verification and production of redeemable tokens [1, 9, 19].

In MicroMint [19], an intricate hashing structure is employed to create redeemable coins. More precisely, a broker creates a coin which satisfies the property that k values all hash to the same result, i.e. a k-way collision. As such, verification is performed by validating that all the k values hash to some result y, $h(x_1) = h(x_2) = ... h(x_k) = y$.

In a similar fashion, Millicent [9] also relies on the creation of broker redeemable tokens, referred to as *scrip*. The scrip tokens comprise two parts: a text scrip and a keyed hash of the scrip using a secret key. The merchant is able to verify a valid scrip by recomputing the hash of the scrip, with a mutually known secret key referred to as the 'scrip secret'. The distribution of the secrets, however, requires additional cryptographic overheads if security is to be maintained.

NetBill [5] provides a comprehensive micropayment system providing an extensive set of security and product delivery services. The scheme uses symmetric cryptography to perform most operations, however there is still the need to perform an initial authentication using public key cryptography for obtaining authentication tickets. This initial authentication is only performed once, and its impact mitigated with a potentially large number of subsequent micropayment transactions verified using symmetric key cryptography.

2.3 Hash Chains

A number of independently devised schemes have an intrinsic hash chain construction [1, 10, 17, 19], and some have augmented these capabilities [14]. These systems are based upon the chained one-time password scheme introduced by Lamport [13]. In reference to micropayments, the key notion is to build a series of payments p_i, for $i = n$ to 0, where the i^{th} payment is related to a previous payment in the form $p_i = h(p_{i+1})$.

The final, or root of the construction is then signed[1], and becomes the first commitment forwarded to the merchant. Subsequent commitments, can be derived from the previous, and hence verified against the hash (or trust) chain to the root of the construction; and the signed commitment. The efficiency gained is that only one signature verification is performed by the merchant on the initial exchange. Remaining commitments are checked for its relationship to the previous iteration, using a computationally efficient primitive, the hash transformation.

A generalisation of the hash chain based schemes is presented in [11], where random numbers form the leaves of a *k-ary* PayTree. The scheme outlined extends the functionality of PayWord providing the ability to spend 'coin certificates' with different merchants (under certain security assumptions). The protocol we present also uses a tree construction, however we restrict the approach to a binary tree under a batch signature paradigm.

2.4 Tools for Signing and Verifying Coins

The concept of batch cryptography was first introduced in [7], and was further developed in [8]. The batch signature scheme outlined, applied RSA by amortising operations over a number of *batched* messages under a binary tree. The fundamental philosophy was that an initial expensive modular exponentiation diminished over many smaller operations. Hence, this diminished complexity was the basis of a performance gain.

In [16] a batch signature scheme is introduced using an alternative binary tree construction. The approach outlined did not impose the restriction that different relatively prime public exponents were required as in [7]. Moreover, only one full sized exponentiation was required to sign a batch of messages. This paper applies the general approach of [16] to devise a new scheme suitable for small payments.

Given the frequency and repetitive nature of micropayments, it is important that verification of commitments or coins be done in an efficient manner. Moreover, many schemes generally avoid the use of exponentiation to verify coins, often at the expense of weaker signatory commitment. We apply the batch signature technique of [16] that enables a number of small coins to be withdrawn in one batch, each bearing an individual bank signature. Withdrawn coins may subsequently be used as small payments with the same, or with different merchants. In particular, when making payments, after some initial exponentiations only hash operations are required to verify the signature on each coin.

To generate a signature for a *batch* of messages, the hash of each message is placed at a leaf node of a binary tree. Each parent node is formed by concatenating each of its child nodes and hashing the result. This continues until the root node T is obtained where the final value is signed using a digital signature scheme. In order to prevent messages representing the internal nodes from having valid signatures, two different hash functions are employed, h_y to hash the leaves and the function h_x to hash the internal nodes. In forming the batch signature of any individual message, we consider

[1] In the various schemes, the signing entity may be the customer or another third party.

the unique path from the node representing the message to the root of the tree; this is found by taking the parent node of each node traversed. The batch residue for that message consists of the sibling nodes of all nodes in this path, together with their direction, which is a single bit denoted L or R. Considering four messages as an example $\{m_1, m_2, m_3, m_4\}$, the residue of m_1 consists of the two nodes $h_y(m_2)$,R and $h_x(h_y(m_3)\|h_y(m_4))$,R while the residue for m_3 consists of the pair $h_y(m_4)$,R and $h_x(h_y(m_1)\|h_y(m_2))$,L.

For message m_i, the residue $m_{\Delta i}$ is the sequence of nodes with their direction (dir), which may be expressed as:

```
c:= h_y (m_i)
for j := 1 to L                                          (1)
        r_j := sibling(c)
        if direction(c) = Left then c := h_x(c || r_j)
        else c := h_x(r_j || c)
endfor
m_Δi = r_1, dir(r_1), r_2, dir(r_2), ..., r_L, dir(r_L)
```

During verification the root node T must be recomputed, using the reside $m_{\Delta i}$, to ensure that purported signature is in fact a signature on message m_i. This is given by:

```
T' := h_y (m_i)
for j := 1 to L                                          (2)
        r_j := sibling(c)
        if direction(r_j) = Left then c := h_x(c || r_j)
        else T' := h_x(r_j || c)
endfor
Output T'
```

The remainder of this paper now expands upon these basic tools for the withdrawal and payment of Microcash coins.

3. Microcash

The following section outlines the Microcash scheme, where the bank efficiently signs a sequence of coins using the binary tree construction outlined in [16]. In order to make a micropayment, the customer must be able to respond to merchant requests for incremental payments, or provide a number of successive and timely payments. In either case, the merchant must be able to make rapid verifications of the submitted payments and the customer must be able to respond with such payments quickly. The following scheme enables both the merchant and customer to make a series of efficient payments in a manner comparable to micropayment schemes.

3.1 Setup and Environment

The bank may employ any signature scheme to sign Microcash coins; a suitable choice would be the RSA signature scheme [20]. The bank publishes its public key. In fact different public keys (for example using different RSA public exponents) should be used by the merchant and bank to represent the number of coins within each Microcash batch. The bank signatures on a message M will be denoted $\sigma(M)$, and there will be a corresponding verification function Ver(M,S) which outputs 1 if S is a valid signature of M, or 0 otherwise.

To demonstrate that the customer took part in the withdrawal we employ the discrete log protocol. Large primes p and q where q divides (p - 1) are chosen and a generator $g \in Z_p^*$, of order q; for the purpose of Microcash p and q could be set to 1024 and 160 bits respectively. Use is also made of a suitable collision free one way function h.

In our cash system each coin is identified by a random token t_i, a proof value b, and its associated signature $\sigma(t_i, b) = (S, m_{\Delta i})$. For the purposes of deposit, a batch of coins is identified by $(t_i, ... t_j, b)$ and signature $\sigma(t_i, ..., t_j, b)$. Using algorithm (2) to recompute the root node, the coins are confirmed valid if Ver(T',S) = 1.

The merchant confirms that the customer presenting the coins took part in the withdrawal protocol when the customer is able to prove knowledge of the secret value w with respect to the proof value b.

3.2 Withdrawal Protocol

The customer, after proving his identity to the bank over an authentication channel, chooses $w \in_R Z_q$, computes $b = g^w \bmod p$, generates a random seed *s* and forms the commitments that may then be used as cash. The customer then proceeds to compute the values $t_1 = h(s)$ and $t_i = h(t_{i-1})$, where the initial seed s is used to compute a sequence of seed values by recursively hashing. This sequence of coins is akin to the chain of commitments generated by other micropayment schemes reliant upon hash chains. When the sequence of seed tokens have been generated the customer then forms a binary tree structure with the values $m_i = h(t_i \| b)$ forming the leaf nodes, recursively hashing until the root node T is formed. The root node is then forwarded to the bank for signing.

The bank initially confirms that the root value T is unique, (this ensures that each batch of coins in circulation is unique, and that double spent coins may be detected later during deposit). The bank then signs the coins using one full signature operation $S = \sigma(T)$, and returns the signed result to the customer.

Upon receipt of the signed tree S, the customer verifiers that the bank has correctly signed the sequence of coins by verifying that the coins bear the bank's signature. Figure 1 illustrates this efficient two move protocol. Note that the bank effort for signing an arbitrary number of coins remains constant.

Customer		**Bank**
Choose s, $w \in_R Z_q$		

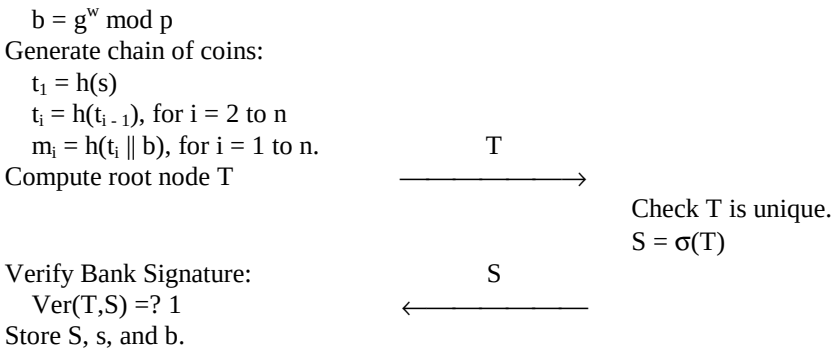

Figure 1 - Withdrawal Protocol

On completion of the withdrawal protocol the customer has at his disposal a sequence of Microcash coins, each one defined by (t_i, b) with corresponding signature $\sigma(t_i, b) = (S, m_{\Delta i})$. The Microcash may now be spent independently, and in off-line manner, with different merchants or with the same merchant. Coins must be spent however, from the last the coin, identified by (t_n, b), first.

3.3 Payment Protocol

The payment protocol consists of a proof phase that involves modular exponentiation, a payment phase that relies only upon efficient hash operations, and an infrequent periodic range check. The initial proof phase may also be used to forward payments from the customer. Fundamentally, the proof phase is used to confirm the banks signature on the coins and that the presenter of the coins took part in the withdrawal. The range check is required to prevent a cheating scenario whereby two merchants collude to deposit the same coins, this is described under the security analysis.

The customer initially establishes contact with the merchant, identified by I_m, and agrees to use a service whereby a number of incremental small payments are required. To commence payment using Microcash coins, the customer identifies the highest sequenced *unused* pair (t_i, b), as the first payment. The associated signature is then selected by identifying the unique residue for (S, m_i), using algorithm (1). The customer forwards to the merchant the first coin (t_i, b), and corresponding signature $\sigma(t_i, b)$. Note that the formation of the unique signature $\sigma(t_i, b)$, may be precomputed. At the same time the customer forwards to the bank a commitment $y = g^u \bmod p$, for a random secret value $u \in_R Z_q$.

The merchant proceeds to verify the bank's signature on the coin, and confirms that the customer did in fact take part in the withdrawal protocol as part of the proof phase of the protocol. To verify the bank signature on a Microcash coin, the merchant recomputes the root value T' using algorithm (2) and checks that $Ver(T', \sigma(T)) = 1$ using the bank's public key. To confirm that the customer took part in withdrawal protocol the merchant also computes the challenge $c = h(dt \| I_m \| t_i \| b)$ using the current the date-time *dt*. The customer, to demonstrate knowledge of secret value w,

forwards response r such that and r = w + cu mod q, to the merchant. This is Schnorr's protocol for proof of knowledge of a discrete logarithm [21].

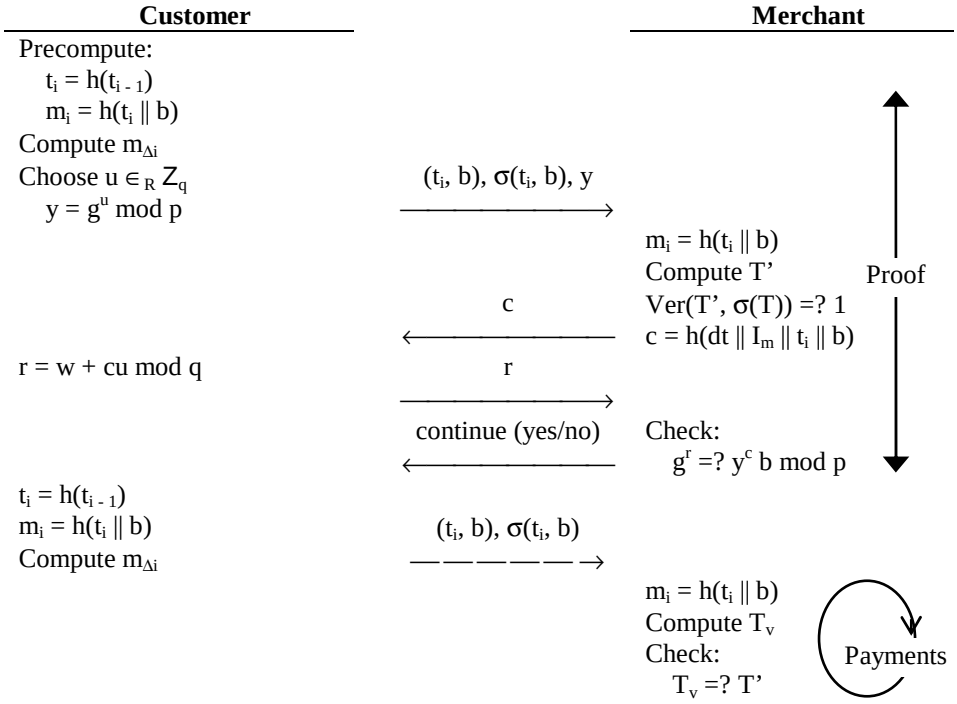

Figure 2 - Payment Protocol

If both the verification and knowledge check, $g^r =? y^c b \bmod p$, hold during the proof phase, then the bank accepts the initial payment and allows the customer to continue using the service. However, subsequent payments do not require any exponentiations during signature verification. Rather, the merchant is only required to recompute T_v, using the signature residue of the next payment (t_i, b) $\sigma(t_i, b)$. Specifically, the merchant checks that signature residue $(m_{\Delta i})$ and coin (t_i, b) can be used to recreate a root node T_v that is equal to T', (note that T' has been computed previously during the proof phase). Thus only three small exponentiations are required during the proof phase, whilst subsequent payments are validated using a number of efficient hash operations. Using this technique, the effort expended to verify the first coin is spread over the total number of coins spent. This procedure is shown in Figure 2.

The customer may decide to compute the challenge for himself (in which case y should be included in the hash), or the merchant may wish to forward this value with some additional information. The customer must also ensure that the random challenge is unique otherwise the ability for colluding merchants to reveal the proof value *w* is given. Furthermore, the response value may be sent together with the next payment, in anticipation of a favourable result during the proof phase.

During the course of customer payments, it is necessary for the merchant to obtain periodic authentication of the range of payments made, see Figure 3. This is achieved by performing, by some periodic instance[2], a similar challenge-response used during the proof phase. The merchant must verify the range check returned by the customer to prevent the customer from successfully double spending; see the security analysis for details of a merchant collusion or customer attacks. When multiple range response pairs are provided during the course of a paying engagement, only the last pair is required for deposit and need be retained by the merchant.

Customer		**Merchant**
	c' ⟵	c' = h(dt ‖ I_m ‖ t_i ‖ b)
Choose u' $\in_R Z_q$		
c' = h(dt ‖ I_m ‖ m_i)	(c', r',y'), (t_i, b),	
y' = $g^{u'}$ mod p	σ(t_i, b) ⟶	Check:
r' = w + c'u' mod q		$g^{r'}$ =? $y'^{\,c'}$ b mod p

Figure 3 - Range Challenge

3.4 Deposit Protocol

During deposit (Figure 4) the merchant is able to obtain funds for the Microcash spent by the customer. Furthermore, the coins spent by a customer may be redeemed in batch, minimising the processing overheads for the bank and merchant during funds capture.

For each coin to be redeemed a transcript of the payment is forwarded to the bank by the merchant. This includes each coin spent (t_i, b) of which there may be multiple instances, the signature associated with the batch of coins σ(t_i ... t_j, b) = (S, $m_{\Delta ij}$), of which there is a single instance, two challenge-response values (one each for the proof phase and last range check), date-time and merchant identity associated with the batch (c, r, y, c', r', y', I_m, dt). Where multiple coins are being deposited the merchant is not required to provide the associated signature for each coin. In fact, the merchant may provide one bank signature by identifying the nodes from each of the batch residues (i.e. nodes of the binary tree) that are sufficient for the bank to verify its own signature, i.e. $m_{\Delta ij}$. (It may be the case, that the bank may wish to perform this pruning activity and accept each signature instance from the merchant).

Merchant	**Bank**
Select signature residue for spent coins:	
Compute $m_{\Delta ij}$	
Forward payment transcripts:	
(t_i, ... t_j, b), σ(t_i ... t_j, b), (y, c, r, y',c', r', I_m, dt) ⟶	For each t_i do:

[2] Perhaps probabalistically; for instance on average every 50 payments.

$m_i = h(t_i \| b)$
Verify Signature:
Compute T'
Ver(T', σ(T)) =? 1
Verify Merchant:
$c =? h(dt \| I_m \| t_i \| b)$
Verify Customer:
$g^r =? y^c$ b mod p
Check Range:
$c' =? h(dt \| I_m \| t_j \| b)$
$g^{r'} =? y'^{c'}$ b mod p

Figure 4 - Deposit Protocol

After confirming that each coin instance (t_i, b) is unique within the database, the bank will then proceed to verify the validity of the coins. This involves checking four proofs. First the bank's signature must be verified, next the bank must confirm that the original (authentic) customer who withdrew the coins was the entity involved in the payment transaction; thirdly the bank also confirms that the merchant depositing the coins was also involved in the payment transaction, and finally the range of coins spent is checked.

In a similar fashion to the customer and merchant, the signature on the deposited coins is verified by the bank recomputing the root node T' and checking that the following holds $T' = S^e \bmod n$. The bank confirms that the merchant took part in the payment protocol, by observing that the merchant identity I_m is associated with the challenge c. To authenticate the customer the bank confirms that the customer can prove knowledge of a representation of y with respect g, i.e. he knows the secret key w. And finally, the bank confirms that the deposited range of coins is correct by checking the range pair supplied by the merchant. This second pair is used to discourage customer or colluding merchants from defrauding the system.

After a certain time, when all the coins have been recaptured, the bank will remove T as being in circulation, and once again will allow a customer to withdraw coins using this batch value.

4. Security Review

We now review the security of the various constructs making up Microcash. We do this by first drawing upon the security analysis of [16], noting that the Merkle assumptions for authentication trees apply to the batch signature scheme. It follows then that each coin may be verified individually by checking the bank signature represented as $\sigma(t_i, b)$. There are however, a number of additional protocol constructions that require a more detailed examination.

4.1 Range Check and a Collusion Attack

The first challenge-response pair confirms to the merchant that the customer is the owner of the coins. Additionally, this authenticates the initial coin spent by the customer, however without some form of final authentication of the finishing point, the range of coins spent is not established. Considering the case that no authentication of the coin range takes place, there are two cheating scenarios:

1. Customer spends coins $t_1...t_j$ with merchant A. Customer then goes to merchant B and starts with t_i, where $i > j$. This means the customer is cheating by double spending $t_i...t_j$.
2. Customer spends coins $t_1...t_{i-1}$ with merchant A. Customer then spends coins from t_i down to t_j with merchant B. Merchant B gives $t_i...t_j$ to merchant A, who may claim payment for $t_1...t_j$.

At deposit time both cases are indistinguishable! To overcome this the customer must periodically provide the merchant with a range proof. During the course of payments a number of these range proofs may be provided, however only the final proof need be retained by the merchant and supplied during deposit.

Depending on the frequency of range checking, the possibility exists for the customer to cheat a small amount. Considering that a cheating customer will be exposed, and that the relative financial gain is too small, defrauding the system is unattractive.

4.2 Double Spent Coins

When the bank detects that a coin (t_i, b) has been previously deposited, the bank first confirms if the coin is within the committed range. Noting that the scheme is not anonymous, the guilty party may determined by observing whether the challenge differs - the same challenge implicates the customer, otherwise the merchant. If however, the coin is beyond the committed range then it is either the customer or colluding merchants who may be guilty. Regardless of the guilty party in this instance, it is the second depositing merchant who does not recover the funds.

To a pair of colluding merchants fraud is not attractive, as detection of the same coin is easy and the bank will not supply the funds. A customer however may be able to cheat successfully, the amount that may be cheated is determined by the merchant by the size of the periodic range check. The relative gain though, will not be worth being exposed as a potential cheat.

5. Efficiency of Scheme

The efficiency of the new scheme is now analysed, paying particular attention to the efficiency of the payment transaction, as it is most important that the payment protocol be performed in a timely and efficient manner. Unless otherwise stated precomputed values (such as y) are ignored, we also ignore the hash operations during the withdrawal and deposit protocols, considering these in more detail during payment.

We then compare our scheme to some other well-known micropayment schemes, demonstrating comparable efficiency, whilst Microcash fields more *cash-like* properties.

We assume that the RSA modulus n is 1024 bits, p and q are 1024 and 160 bits respectively, and the hash function produces digests of 160 bits in length. When considering the exponentiations we note that the discrete log exponentiations using the exponent q are 6.4 (1024/160) times cheaper [12]. Although we set the size of parameters to those required of full cash systems, in practice these may be lower given the nature of the small financial amount of each coin.

During withdrawal the merchant is only required to perform one RSA modular exponentiation to sign an arbitrary number of coins. The customer performs one cheaper (factor of 6.4) modular exponentiations to compute b, and the more efficient RSA verification exponentiation to verify the bank's signature on all coins withdrawn. A total of 148 bytes is transmitted between the customer and the merchant. The customer is only required to provide storage for 276 bytes; for values S, s, and b.

We consider payment by assessing the hash operations performed during each payment. An initial modular multiplication/addition is required during the proof, and an arbitrary number of additional modular multiplications/additions are required for range checking (perhaps 1 for every 50 payments). The customer is required to perform at most O(1.5n) hash operations to create the coin and its signature for each payment. This can be reduced to O(n/2) operations if the child node of the root node, not in the path of the coin, is stored (at a cost of 20 bytes storage). Furthermore, much of the work here can be precomputed, and optimised significantly further if more intermediate tree nodes are stored. The merchant processing involves one RSA verification and two discrete log modular exponentiation during the proof phase. In addition to the proof, the recreation of the binary tree costing O(log n) hash operations[3], using the supplied coin and signature residue, is required for each coin payment and a probabilistic range verification is required. As such, we generalise the cost of verification to O(log n) hash operations, with the effort of three modular exponentiations amortised over a large number of successive efficient payments. The additional range verifications contribute an additional 2 modular exponentiations for every 20-50 payments, where the specific frequency is determined by the merchant. As such, there is an efficiency versus security *trade-off* that is determined by the merchant.

During the deposit transaction the merchant requires no specific processing other than a selection process to identify the required signature residue nodes to be supplied for the coin(s) to be deposited. Similar to the merchant, the bank performs four discrete log exponentiations and one RSA verification. Additionally, approximately O(log n) hash operations are required to build the binary tree. This of course can be performed off-line some time after the payments have been made, perhaps at the end of each billing day.

[3] For example, if the batch consists of 1000 coins only 11 hash operations are required for verification.

5.1 Comparative Analysis

We now compare Microcash to well known micropayment schemes. Specifically, we shall contrast to the schemes of PayWord [19], Millicent [9], and MicroMint [19]. In our comparison we will assume that RSA signatures are used for the bank's signature on coins. We first briefly noting the properties of Microcash:

- Coins do not have an expirary period,
- Coins may be spent at different merchants,
- Microcash is a debit based scheme,
- Microcash is an off-line scheme,
- No wastage of computation activities (e.g. stick wastage),
- No protocol is necessary to redeem unspent coins,
- Coins may not be stolen or replayed, and
- No additional security mechanisms are required.

The PayWord scheme relies on the creation of a sequence of commitments which are vendor specific (cannot be used with any other vendor). During commitment generation (equivalent to withdrawal of our scheme) the customer creates a sequence of commitments and signs the root, one RSA modular exponentiation. When making payments the merchant verifies the customer's signature on the PayWord sequence, and verifies the brokers signature on the customers certificate: two RSA verifications. To confirm each payment only one hash operation is require; furthermore each payment is 20 bytes in length. Our scheme is more expensive in computation during payment, however we offer additional functionality:

- PayWord coins are merchant specific, whilst Microcash coins may be spent with any merchant.
- PayWords expire at the end of each day.
- PayWord is primarily intended as a credit based scheme, a debit based version would expose signed PayWords to theft.

Millicent [9] relies on the creation of vendor specific scrip, obtained from a broker. The scrip tokens comprise two parts: a text scrip and a keyed hash of the scrip using a symmetric 'scrip secret' key known by the merchant and customer alike. The distribution of the secret key, requires additional cryptographic overheads if security is to be maintained decreasing the efficiency of the scheme. Additionally, Millicent requires that the broker is on-line during the payment transaction:

- Millicent coins are only valid for a specific vendor, Microcash coins may be spent with any merchant.
- Unspent Millicent coins must be presented at the bank in a coin return protocol to obtain unspent funds.
- Millicent coins have an expiration date, and customer must renew or cash in unused scrip; no such expiration applies to Microcash coins.
- Millicent is an on-line protocol.

The MicroMint scheme is similar to Microcash in that coins are generated which may be spent at any merchant. MicroMint coins are represented as a k-way hash function collision, a 4-way collision is suggested as reasonable. To create coins, equivalent to our withdrawal, the broker must engage in extensive computation of

hash transformations to find 4-way collisions. For example, to create 2^{30} coins, approximately 2^{54} hash computations are required; and 128 gigabytes of storage for 2^{30} coins. Each coin expires at the end of the month, so the minting process must be done monthly. Conversely, an arbitrary number of coins may be signed in each Microcash withdrawal, hence no lengthy precomputation is required and no expiry time need be observed. During payment a MicroMint coin is forwarded to the merchant, who performs k hash operations to check if the coin is good, i.e. k collisions; the bank performs similar processing during deposit. To prevent theft of MicroMint coins additional encryption primitives such as private and symmetric key technologies are suggested, or user-specific coins may be introduced; both techniques serve to decrease efficiency. Summarising the properties of the scheme:

- MicroMint coins expire at the end of each month.
- Without additional security MicroMint coins may be stolen and replayed, whilst Microcash coins cannot be stolen and used by other parties.
- MicroMint requires a lengthy precomputation phase.
- MicroMint coins may be stolen, additional cryptographic mechanisms are required to protect the scheme from theft.
- Signatures are not used in MicroMint thus cheaters cannot be pursued in court.
- Unused MicroMint coins are no longer tenable and constitute non-productive processing.

6. Conclusions

This paper has presented a practical off-line electronic cash system for conducting a number of efficient small payments. The scheme offers a number of cash-like properties, such as off-line coin validation, whilst maintaining the efficiency required of small successive payments. We apply batch cryptographic techniques in a novel way to provide this efficiency, in a way that produces a number of coins, each bearing their own signature from the bank. A comparison to some micropayment systems demonstrates that the efficiency of our scheme comparable; this is achieved whilst providing a number of additional cash-like properties.

6.1 Extensions

Since the merchant may deposit coins in batch manner, if follows that a similar protocol may be adopted by the customer and merchant that enables the payment of multiple coins during one transaction. Hence, payments can be made one coin at a time, or can be made with multiple coins spent at a time.

6.2 Further Work

Range checking during payment discourages the collusion attack identified. This mechanism will incur additional computational costs and may become excessive

where greater security requirements prevail - the demand for a higher probabilistic frequency check. Therefore, we suggest that the elimination, or improvement, of the range check remains as further work on this scheme.

References

1. R. Anderson, C. Manifavas, C. Sutherland, NetCard — A Practical Electronic Cash System, Proceedings of 4th Cambridge Workshop on Security Protocols, Springer-Verlag, 1996.
2. M. Bellare, J. A. Garay, R. Hauser, A. Herzber, H. Krawczyk, M. Steiner, G. Tsudik, M. Waidner, *i*KP - A Family of Secure Electronic Payment Protocols, First Usenix Workshop on Electronic Commerce, pp89-106, 1995.
3. S. Brands, Off-Line Electronic Cash Based on Secret-Key Certificates, Proceedings of the Second International Symposium of Latin American Theoretical Informatics - LATIN ' 95, April 1995.
4. D. Chaum, A. Fiat, M. Naor, Untraceable Electronic Cash, Advances in Cryptology - Crypto '88, Springer-Verlag, Vol. 403, pp319-327, 1988.
5. B. Cox, J. D. Tygar, M. Sirbu, NetBill Security and Transaction Protocol, Proceedings of the First Usenix Workshop on Electronic Commerce, 1995.
6. D. Eastlake, B. Boesch, S. Crocker, M. Yesil, CyberCash Credit Card Protocol, RFC 1898, Version 0.8, February 1996.
7. A. Fiat, Batch RSA, Proceedings of Crypto '89, Springer-Verlag, Vol. 435, pp175-185, 1990.
8. A. Fiat, Batch RSA, Journal of Cryptology, Vol 10, pp75-88, 1997.
9. S. Glassman, M. Manasse, M. Abadi, P. Gauthier, P. Sobalvarro, The Millicent protocol for inexpensive electronic commerce, Proceedings of Fourth International World Wide Web Conference, pp603-618, O'Reilly, December 1995.
10. R. Hauser, M. Steiner, M. Waidner, Micro-Payments based on *i*KP, SECURICOM '96, 1996.
11. C. Jutla, M. Yung, Paytree: "amortized signature" for flexible micropayments, Proceedings of 2nd USENIX Workshop on Electronic Commerce, pp213-21,1996.
12. N. Koblitz, A course in number theory and cryptography, Graduate Texts in Mathematics, Springer-Verlag, Vol. 114, 1987.
13. L. Lamport, Password Authentication with Insecure Communication, Communications of the ACM, No. 24, pp770-772, 1981.
14. Y. Mu, V. Varadharajan, Y. Lin, New Micropayment Schemes Based on PayWords, Proceedings of ACISP '97, Springer Verlag, LNCS, Vol. 1270, pp283-293, 1997.
15. T. Okamoto, K. Ohta, Universal Electronic Cash, Advances in Cryptology - CRYPTO '91, Springer-Verlag, Volume 576, pp324-337, 1991.
16. C. Pavlovski, C. Boyd, Efficient Batch Signature Generation Using Tree Structures, Proceedings of CrypTEC '99, Hong Kong, 1999.
17. T. Pedersen, Electronic Payments of Small Amounts, Proceedings of International Workshop on Security Protocols, pp59-68, 1996.
18. R. Rivest, Electronic Lottery Tickers as Micropayments, Proceedings of Financial Cryptography '97, 1997.
19. R. Rivest, A. Shamir, Payword and MicroMint: Two Simple Micropayment Schemes, Proceedings of International Workshop on Security Protocols, pp69-87, 1996.

20. R. Rivest, A. Shamir, L. Adleman, A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, Communications of the ACM, Volume 21, Number 2, pp120-126, February 1978.
21. C.P. Schnorr, Efficient Signature Generation by Smart Cards, Journal of Cryptology, 4, pp.161-174, 1991.
22. SET Specification, Book 1: Business Description, Mastercard & Visa, Version 1.0, 31 May 1997.
23. D. Wheeler, Transactions Using Bets, Security Protocols, Lecture Notes In Computer Science, 1189, pp89-92, 1996.