Detachable Electronic Coins

Chris Pavlovski, Colin Boyd, and Ernest Foo

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

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

Abstract. *We propose a non-divisible electronic cash system that combines the restrictive blind signature technique and a batch signature scheme based upon the binary tree structure. Contemporary electronic cash solutions rely on the divisibility paradigm to achieve significant computational savings, and such techniques are considered to be the most efficient. We demonstrate an alternative approach that employs batch cryptography to withdraw and spend multiple coins. This approach provides a set of alternative primitives for an electronic cash system that possesses comparable efficiency to the most efficient divisible based schemes.*

1. Introduction

Electronic cash is the virtual manifestation of physical cash, endeavoring to possess similar properties to physical cash including portability, transferability, and anonymity. Additionally, electronic cash is subject to the same attempts at fraudulent use as physical cash.

Physical cash systems essentially involve three entities: the merchant, customer and bank. In much the same way, electronic cash systems involve these entities; however, the presence of physical notes and coins is replaced by a bitstring that may be easily copied and reused. To masquerade bit strings as cash, a number of security properties must be satisfied. These include identity detection of a double spending party, the ability to conduct payments without being traced, and difficulty in forgery. Whilst there are a number of other desirable features, such as prior restraint of double spending, these properties appear to be the primary constituents of many electronic cash systems.

Electronic cash propagates through society under three prominent transaction analogies; these are the withdrawal, payment, and deposit transactions. The customer, after identification, establishes an account with a bank, and is then able to obtain electronic notes and coins from the bank by way of the withdrawal transaction. To purchase items the customer presents the electronic cash to a merchant during the payment transaction. And finally, a merchant is able to redeem hard currency for the electronic cash by conducting the deposit transaction with the bank. A key feature of any proposed electronic cash system is the ability to conduct payment in an off-line manner, i.e. without the involvement of a third party, particularly the bank. Furthermore, both the withdrawal and payment transactions should be efficient, this is quite crucial as merchants and banks represent a central processing point in a distributed cash environment.

Despite the intense research of this area [3, 8, 13, 15, 16, 14, 5], electronic cash is yet to make significant inroads into the commercial sector. A major reason for this predicament is the inability of commercial entities to conduct electronic cash transactions in an efficient manner. In this paper we propose a new electronic cash system that provides efficient withdrawal and payment, and exhibits computational advantages for deposit. The scheme differs considerably from other electronic cash schemes in that it does not rely on a divisibility for improving efficiency when dealing with multiple coins. Rather, we demonstrate an alternative approach where multiple coins are withdrawn and spent using batch protocols. We demonstrate that the efficiency exceeds previous schemes [3, 14] and is comparable to the most efficient cash protocols that employ a divisible structure [5]. The fundamental protocols are based upon the restrictive blind signature schemes introduced by Brands [3] and the batch protocols proposed in [17, 2, 21].

1.1 Contribution

An electronic cash scheme [1] that was recently proposed by us employs the concepts of batch cryptography to improve the computational efficiency of withdrawing non-divisible coins. This scheme has similarities to previous work of Tsiounis [19]. A significant drawback of the scheme is that the signature size becomes unacceptably large for an increasing batch size n, increasing as O(n). For example, when withdrawing 1,000 coins in batch the associated signature, which is part of any one coin, is 20 Kbytes. We avoid this linear size increase by applying the batch signature scheme of [17] using a binary tree structure. Under our proposed new scheme, the corresponding signature residue now becomes 200 bytes!

In this paper, we view our main contribution as proposing the first practical electronic cash system that provides an efficient alternative to divisible based cash schemes. The efficiency gains are by way of improvements in computation, communications and storage using novel batch signature and verification techniques based upon a binary tree structure. Specifically, we solve the problem of an increasing signature size for batch coins, intrinsic to [1], by reducing it to O(log n). Furthermore, we introduce several additional efficiency improvements in computation and storage.

We demonstrate that our scheme is more efficient than other electronic coin schemes [3, 14] and observe the scheme to be comparable to the most efficient divisible based systems exhibiting similar e-cash properties [5].

1.2 Organisation of Paper

The next section provides background techniques used in the electronic cash scheme we present, including details of the new concepts applied in our scheme. This is followed in section 3 by a detailed description of the new cash scheme. We then assess the security and analyse the improvements in efficiency.

2. Background

We describe the various constructions that provide the foundations of the new electronic cash system. Commencing with Brands' restrictive blind signatures and representation problems, we propose a Schnorr batch tree signature structure for improved efficiency.

2.1 Restrictive Blind Signatures

Brands [3] introduced a restrictive blind signature that extends the protocol of Chaum and Pedersen [7]. The term *restrictive* is evident in the protocol as the customer is restricted to a particular form when blinding coins, more precisely he must encode his identity. In the case of [3], the customer *must* know the representation $(u_1, 1)$ of A with respect to (g_1, g_2) (see below for a discussion of the representation problem). This restriction (on the blinding of the internal structure of coins) will enforce the ability to reveal the identity of a customer if there is an attempt to double spend. Of course if the restricted form is not adhered to, the customer does not end up with a valid signature on the blinded coins.

As in all the subsequent equations, two prime numbers p and q must be used, where q is a prime factor of p - 1, and an element g of order q. All computations take place modulo p unless otherwise stated. The public and private keys are x and h, where $0 < x < q$, and $h = g^x \bmod p$. The restrictive blind signature scheme is as follows.

Stage 1. The signer generates a random values $w \in Z_q$ and sends $z = m^x$, $a = g^w$, and $b = m^w$ to the verifier.

Stage 2. The receiver generates blinding invariants s, u, and $v \in Z_q$ and computes the following (returning c to the signer):

$$m' = m^s$$
$$a' = a^u g^v$$
$$b' = b^{su} m^v$$
$$z' = z^s$$

$c' = H(m', z', a', b')$, and returns the blinded challenge $c = c'/u \bmod q$ to the signer.

Stage 3. The signer computes the response $r = cx + w \bmod q$ to the verifier.

Stage 4. The receiver accepts the signature as valid if the following hold:

$$g^r = a \; h^c$$
$$m'^{\,r} = b \; z^c$$

The receiver computes r' = ru + v mod q, then ends up with the message m' blindly signed by the signer with corresponding signature (z', a', b', r'), which may be verified by others.

2.2 Representation problem

The representation problem, introduced by Chaum *et al.* [4], underpins the protocols of Brands' cash system. Simply put, the representation problem is: given a value h and the pair (g_1, g_2), find a pair (x_1, x_2) such that $h = g_1^{x1} \, g_2^{x2}$. In fact any number of g_i values can be used, and it can be seen that the representation problem is a generalisation of the discrete log problem which is the case of a single g value.

A coin in Brands' scheme [3] is a pair (A,B) and during the payment protocol the customer must prove knowledge of B's representation. This is achieved by proving knowledge of (x_1, x_2), showing that he did in fact take part in withdrawing the coins. Additionally, under the restrictive blind signature technique the customer must know a representation of A with respect to (g_1, g_2). The following protocol demonstrates how one may prove knowledge of a representation (x_1, x_2) of h with respect to (g_1, g_2).

Stage 1. The prover generates random values $w_1, w_2 \in Z_q$ and sends $z = g_1^{w1} \, g_2^{w2}$ to the verifier.

Stage 2. The verifier responds with a random challenge $c \in Z_q$.

Stage 3. The prover computes responses $r_1 = w_1 + cx_1$ mod q, and $r_2 = w_2 + cx_2$ mod q, and forwards them to verifier.

Stage 4. The verifier accepts that the prover knows a representation, if the following holds:

$$z \, h^c = g_1^{r1} \, g_2^{r2}$$

2.3 Batch Cryptography

The concept of batch cryptography to improve efficiency was first introduced by Fiat [9], through a technique that combined several messages for batch signing under RSA. The concept was further applied to Diffie-Hellman key agreement [20] and to DSA batch verification [12]. Further work concerning batch verification has been discussed for both the Schnorr and RSA digital signature schemes [21]. More recently it has been demonstrated [2] that some batch verifiers are not true verifications and that a weaker property called *screening* applies.

We next describe an alternative technique for signing a batch of messages, the objective of which is to reduce the size of the generated signature. In this paper, use is made of a binary tree structure to combine several messages into one batch for signing.

2.4 Extensions to Previous Schemes

Our major extension involves a Schnorr variant of a batch signature scheme that employs a binary tree structure. A detailed description of this technique may be found in a previous paper [17]. We also discuss several improvements in the efficiency of the batch representation check. Before describing the batch tree protocol with the Schnorr signature scheme [18], we must briefly outline the batch signature scheme for an arbitrary number of messages.

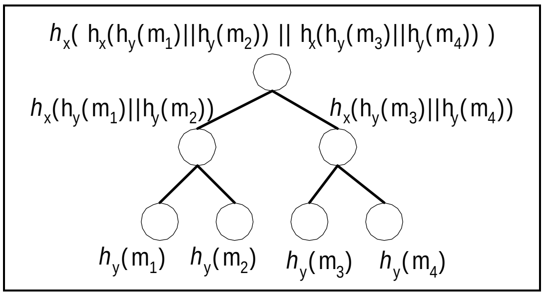

Figure 1 - Binary Tree of Messages

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 the Schnorr signature scheme. In order to prevent messages representing the internal nodes from having valid signature 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 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. An example with four messages is shown in Figure 1. In this example the residue of m_1 consists of the two nodes $(h_y(m_2), R)$ and $(h_x(h_y(m_3)\| h_y(m_3)), R)$ while the residue for m_3 consists of the pair $(h_y(m_4), R)$ and $(h_x(h_y(m_1)\| h_y(m_2)), R)$.

More formally, for message m_i the residue $T_{res\ i}$, the sequence of nodes with their direction (dir), may be expressed as:

$$
\begin{array}{l}
c := h_y\ (m_i) \\
\text{for } j := 1 \text{ to } L \\
\quad\quad r_i := \text{sibling}(c) \\
\quad\quad \text{if direction}(c) = \text{Left then } c := h_x(c \,\|\, r_i) \\
\quad\quad \text{else } c := h_x(r_i \,\|\, c) \\
\text{endfor} \\
T_{res\ i} = r_1, \text{dir}(r_1), r_2, \text{dir}(r_2), \ldots, r_L, \text{dir}(r_L)
\end{array} \quad \mathbf{(1)}
$$

During verification the root node T must be recomputed to ensure that purported signature is in fact a signature on message m_i, this is given by:

$T' := h_v\ (m_i)$ **(2)**
for j := 1 to L
    r_i := sibling(c)
    if direction(r_i) = Left then $c := h_x(c \parallel r_i)$
    else $T' := h_x(r_i \parallel c)$
endfor
Output T'

2.4.1 Batched Schnorr Signature Scheme

The Schnorr signature scheme [18] has public and private keys x and h, where $0 < x < q$, and $h = g^x \bmod p$. To sign message *m* with the private key x, outputting signature (c, r), the following steps are performed:

1. $a = g^w \bmod p$, where $1 < w < q - 1$.
2. $c = h(m \parallel a)$
3. $r = w + xc \bmod q$

To verify the signature (c, r), compute $a' = g^r \cdot h^c \bmod p$, and check that $c = h(m \parallel a')$.

Adapting the protocol, we are able to generate *n* signatures (C, r, $T_{res\ i}$) on a sequence of *n* messages m_i, for i =1 to *n*, in the following manner:

1. $a = g^w \bmod p$, where $1 < w < q - 1$.
2. Assign messages as leaves of the tree and compute root node T.
3. $C = h(T \parallel a)$
4. $r = w + xC \bmod q$
5. Using algorithm (1) compute $T_{res\ i}$, for i = 1 to *n*.

Any entity is able to verify the individual signature (C, r, $T_{res\ i}$), on an associated message m_i. To verify compute $a' = g^r \cdot h^C \bmod p$, compute T' using algorithm (2), and check that $C = h(T' \parallel a')$. The receiving entity is able to independently verify the signature of the sender on message m_i. The security of this scheme is proven in [17].

Examining Brands' protocol [3], we rely on the separation of m into A and B, with the identity encoded within A to detect double spending. We replace B however, with a sequence of B_i values, one for each coin, signed with the batch signature tree structure. We do this in way that offers *detachability*, rather than divisibility for the partitioning of coins.

2.4.2 Representation Screening and Improved Batch Verifiers

We introduce extensions to the payment and deposit transactions of [1], improving the efficiency for the merchant and bank respectively. In particular, we discuss whether alternative, more efficient, batch representation checks may be applied (namely those of Bellare *et al*. [2]). We show (see Appendix A) that an improved batch verifier, the Bucket test, may be useful to the merchant during payment, and that *representation screening* may be practiced by the bank during deposit, improving the efficiency of the deposit transaction.

3. Detachable Coin Scheme

The customer selects a secret number $u_1 \in_R \mathbb{Z}_q$, computes the account number $I_c = g_1^{u_1}$, and forwards this to the bank. The bank stores the customer's details, unique account number I_c, and publishes the bank's public key $h = g^x$, for private key x. The bank also publishes g_2^x, permitting later computation of $z = (I_c g_2)^x$ by the customer. Different generators for g_2 may be used to identify both the denomination and the number of coins to be withdrawn in the cash system. In this environment, we use g_2 to indicate the number of coins that are withdrawn at any one time. This is later used by both the merchant and bank to confirm that a correct number of coins have been withdrawn and that there are not counterfeits.

In our cash system each coin is identified by (A, B_i, i), $i \in \{1\ldots n\}$, and its signature $\sigma(A, B_i) = (T_{res\ ij}, z', a', b', r')$. A batch of coins is identified by $(A, B_j, \ldots B_k, i_j \ldots i_k)$ and signature $\sigma(A, B_j, \ldots B_k)$. The coins are valid when the following equations are satisfied (with suitable hash function $\mathcal{H}$):

$$g^{r'} = a \cdot h^{\mathcal{H}(A, T', z', a', b')}$$
$$A^{r'} = b \cdot z^{\mathcal{H}(A, T', z', a', b')}$$

The merchant can be sure that the customer presenting the coins took part in the withdrawal protocol when the customer is able to prove knowledge of the coin's secret representation pair (x1, x2) with respect to (g_1, g_2).

3.2 Withdrawal Protocol

The withdrawal protocol enables the customer to receive a signature on a batch of coins, in a way that enforces the encoding of the customer's identity for each coin in the batch. For simplicity, we follow the notation of Brands [3] and derivations thereof [1]. In the interests of efficiency, we also assume that the customer precomputes the electronic coin tokens B_i prior to interaction with the bank; we describe this step first.

The customer manufactures a sequence of *n* electronic coin tokens, the number *n* determined by the bank's advertised generator g_2, encoding his identity and maintaining a representation with respect to (g_1, g_2). Choosing blinding invariant s, and initial secret representation values $x1_1$, $x2_2 \in_R \mathbb{Z}_q$, remaining values are computed as $x1_i = h(x1_{i-1})$ and $x2_i = h(x2_{i-1})$ for i = 1 to n. The customer builds a representation of each coin by computing $A = (I_c g_2)^s$, $B_i = (g_1^{x1i}\, g_2^{x2i})$. Each coin (A, B_i) encodes the identity I_c, and is known to the customer by the unique representations $(u_1 s, s)$ and $(x1_i, x2_i)$ with respect to (g_1, g_2). The coins are then marshalled into a tree to build the unique structure representing the batch of coins. Commencing with a hash of each coin token B_i assigned to the leaf nodes, the tree is built until the root node T, and unique presentation of the coin batch is constructed. At this point the customer has encoded the identity into several batch coins for an interactive signature from the bank, of which the representation is known. The internal node values are not discarded as they may be used later during payment when forming the signature for the unique selection of coins.

Before commencing the on-line phase of the withdrawal protocol, it is assumed that the customer has proven his identity to the bank over an authenticated channel. The bank chooses a secret value $w \in Z_q$, and sends to the customer (blinded) values $a = g^w$, $b = m^w$. The customer, using the previously manufactured sequence of coins, now performs a number of steps that involve generating the batch of blinded coins for signatory endorsement. In order to hide the true identity of the customer from the bank during signing, the customer chooses additional blinding invariants u and v $\in Z_q$, computes new values $a' = a^u g^v$, $b' = b^{su} A^v$, and $z' = z^s$. A challenge $c' = \mathcal{H}(A, T, z', a', b')$ is formed and the blinded translation $c = c'/u$ is then forwarded to the bank.

The bank responds to the challenge with $r = cx + w \bmod q$ and deducts the customer's account by the amount withdrawn. The customer confirms the presence of a correctly formed signature by checking $g^r = a\, h^c$ and $(I_c g_2)^r = b\, z^c$. The signature on the batch of coins now becomes the tuple $(T_{resi}, z', a', b', r')$, where r' is derived as $r' = ru + v \bmod q$. The following steps summarise the protocol for withdrawal, (where stage 1 may be performed prior to the on-line phase consisting of stages 2 onwards):

Stage 1. The customer randomly chooses s, $x1_1$, $x2_1 \in_R \mathbb{Z}_q$, encoding the identity $A = (I_c g_2)^s$, and constructing a unique representation of each coin as $B_i = (g_1^{x1i} g_2^{x2i})$, where $x1_i = h(x1_{i-1})$ and $x2_i = h(x2_{i-1})$. The coins, forming the leaf nodes, are now marshalled into a binary tree applying function h_y to leaf nodes and h_x to internal nodes until the root node T is constructed.

Stage 2. The bank chooses $w \in Z_q$, computes and sends $a = g^w$, $b = m^w$ to the customer.

Stage 3. Choosing blinding invariants u and v $\in Z_q$, new values of a, b and z are computed as $a' = a^u g^v$, $b' = b^{su} A^v$, $z' = z^s$. The value $c' = \mathcal{H}(A, T, z', a', b')$ is computed with blinded challenge $c = c'/u \bmod q$ returned to the bank.

Stage 4. The bank returns $r = cx + w \bmod q$ to the customer.

Stage 5. The verifier accepts the signature as valid if the following hold:

$$g^r = a\ h^c$$
$$(I_c g_2)^r = b\, z^c$$

If the verification holds the customer computes $r' = ru + v \bmod q$. Each coin forming the batch may be individually identified by $\{A, B_i, i, T_{res\,i}, z', a', b', r'\}$, where $\{T_{res\,i}, z', a', b', r'\}$ is the signature on the unique pair (A, B_i). Recall that $T_{res\,i}$ is composed of each highest level node not in the path of the leaf node identified by B_i. Each coin may be detached from the tree and individually spent with different merchants. Multiple coins may also be spent with the same merchant during the same protocol exchange. For instance, three coins may be identified with the following bitstring sequence $\{A, B_1, B_2, B_3, i_1, i_2, i_3, T_{res\,123}, z', a', b', r'\}$.

The customer has encoded the identity within each pair (A, B_i), such that double spending will reveal the identity I_c of the customer. As in the scheme of [1], the sequence of batch coins may be linked between different transactions; however, coins from different batches cannot be correlated. This linkability does not reveal the customer's identity, rather it allows the bank to identify which transactions are from the same customer (or rather, which batch of coins).

3.3 Payment Protocol

When a customer wishes to spend valid coin(s) at a merchant shop, the batch sequence (A, B_j, ... B_k, i_j ... i_k) and corresponding signature σ(A, B_j ... B_k) are forwarded to the merchant. Note that the number of coins to be spent is defined by the number of B constructions forwarded. The coins may be spent individually or in batch manner, and may be forwarded in no particular order. For each set of coins to be spent in one transaction, the unique batch residue $T_{res\, jk}$ (for leaf nodes j to k) must be constructed by selecting the appropriate intermediate nodes from the binary tree. Note that it is not necessary to recompute these values as they may be stored after the execution of algorithm (1).

On receipt of the coins, the merchant computes the challenge $d = \mathcal{H}(A, B_i \ldots B_j, i_j \ldots i_k, I_m, t)$, where t represents a date/time stamp and I_m is the identity of the merchant.

Using the customers' secret value u_1, and blinding invariant s, the customer demonstrates knowledge of the coins representation (x1, x2) with respect to (g_1, g_2) by computing responses $r1_i$ and $r2_i$ for each coin *i* to be spent in the transaction:

$$r1_i = du_1s + x1_i, \ldots, r1_j = du_1s + x1_j$$
$$r2_i = ds + x2_i, \ldots, r2_j = ds + x2_j$$

On receipt of the response values, the merchant proceeds to verify the bank's signature on the coins and checks that the customer knows a representation of the coins with respect to (g_1, g_2).

The bank's signatures on the blinded coins are checked by computing T' using algorithm (2), and then confirming that $g^{r'} = a' \,.\, h^{\mathcal{H}(A, T', z', a', b')}$ and $A^{r'} = b' \,.\, z'^{\mathcal{H}(A, T', z', a', b')}$. The merchant confirms that the customer knows representation of the coins by first selecting the optimal batch verifier (the bucket test is selected as the batch size grows beyond 200). Assuming for the moment that the batch size is small, random security values w_j ... w_k are chosen and the following is checked (3) $\prod_{i=j}^{k} g_1^{r1_k w_k} g_2^{r2_i w_i} = \prod_{i=j}^{k} A^{dw_i} B_i^{w_k}$. The values w_j through w_k are required to prevent the customer from choosing, with high probability, r values that would satisfy equation (3).

If both the verification and representation check succeed, the merchant accepts the coins as valid and may then forward the goods to the customer. Coins that are detected to be from the same batch may be stored together, optimising the storage consumed by coins. In a similar manner, the customer can remove the coin token values B_i from storage. Summarising the payment protocol, we view the following steps to spend a coin sequence (B_j ... B_k) from the same batch:

Stage 1. The customer creates the unique signature residue $T_{res\ jk}$, under algorithm (1), for coins to be spent and sends {A, B_j, ... B_k, i_j ... i_k, σ(A, B_j, ..., B_k)}to the merchant.

Stage 2. The merchant computes and returns challenge d = $\mathcal{H}$(A, B_j ... B_k, i_j ... i_k, I_m, t).

Stage 3. The customer computes responses and forwards to the merchant:

$$r_{1j} = du_1 s + x1_j ,..., r_{1k} = du_1 s + x1_k$$

$$r_{2j} = ds + x2_j \ ... \ r_{2k} = ds + x2_k$$

Stage 4. The merchant selects the optimal batch verifier. Assuming that the batch size is small, compute T' using algorithm (2), choose w_j, ..., $w_k \in Z_q$, and accept the coins as valid if the both the signature verification and representation check hold:

$$g^{r'} = a' \,.\, h^{\mathcal{H}(A,\, T,\, z',\, a',\, b')}$$

$$A^{r'} = b' \,.\, z'^{\mathcal{H}(A,\, T,\, z',\, a',\, b')}$$

$$\prod_{i=j}^{k} g_1^{r1_k w_k} g_2^{r2_i w_i} = \prod_{i=j}^{k} A^{dw_i} B_i^{w_k}$$

The merchant stores the spent coins and transaction details consisting of {A, B_j, ..., B_k, i_j ... i_k, σ(A, B_j, ..., B_k), t, r_{1j}, ... r_{1k} , r_{2j}, ... r_{2k}}. Under this scheme, coins spent from the same batch may be linked together. This however, may be controlled by the customer by increasing the number of distinct batch withdrawals, as different batches cannot be linked.

3.4 Deposit Protocol

The merchant is able to redeem hard currency from the electronic cash during the deposit transaction with the bank. A transcript of the payment, including the merchant's identity, is forwarded to the bank. Similar to the merchant, the bank performs a signature verification, but performs a more efficient representation check by screening the coins' representation. This is completed by first recomputing the merchant challenge d = $\mathcal{H}$(A, B_j ... B_k, i_j ... i_k, I_m, t), and then confirming that the signature and representation equations hold.

Stage 1. The merchant sends coins (A, B_j, ... B_k, i_j ... i_k) σ(A, B_j ... B_k), representation responses (r_{1j}, ..., r_{1k}, r_{2j}, ..., r_{2k}) and merchant particulars (t, I_m) to the bank.

Stage 2. The bank recomputes the merchants challenge d' = $\mathcal{H}$(A, B_j ... B_k, i_j ... i_k, I_m, t).

Stage 3. The bank reconstructs the root node T using algorithm (2), verifies its own signature on the coin, and screens the representation of the coins:

$$g^{r'} = a' \,.\, h^{\mathcal{H}(A,\, T,\, z',\, a',\, b')}$$

$$A^{r'} = b' \,.\, z'^{\,\mathcal{H}(A,\, T,\, z',\, a',\, b')}$$

$$\prod_{i=j}^{k} g_1^{r1_i} g_2^{r2_i} = \prod_{i=j}^{k} A^d B_i$$

If either the representation screen or coin verification fails then the bank rejects the coins. Before crediting the merchant's account, the bank checks that the coins have not been deposited previously. This involves searching a database for the same coins, each one uniquely identified by (A, B_j) for $j = 1$ to n (note that each batch may be identified by A). If the coins have not been previously spent then the bank credits the merchant's account and stores the values $(A, B_i, \ldots , B_n, i_j \ldots i_k)$, $(r_{1j}, \ldots, r_{1k}, r_{2j}, \ldots, r_{2k})$, t and I_m.

It should be noted that the general attack on RSA screening [6] does not apply to representation screening as duplicate messages constitute double spent coins and may be easily detected. It follows then that the pruning step of [2] is not necessary.

3.4.1 Revealing Identity of Double Spender

In the event that the bank detects doubling spending, either the merchant is attempting to deposit the same coin twice or the customer has double spent the same coin. As in Brands' original scheme [3], it is now straightforward to identify the perpetrator. If the value t is the same as the value in the database then the merchant is guilty, otherwise the customer has double spent the coin. The bank has at its disposal knowledge of the two relationships $r_{1i} = dus + x_{1i}$, $r'_{1i} = d'us + x_{1i}$, and $r_{2i} = ds + x_{2i}$, $r'_{2i} = d's + x_{2i}$. Using $I_c = g_1^{(r_{1i} - r'_{1i}) / (r_{2i} - r'_{2i})}$, the bank is able to derive u, which proves that the customer has double spent, and is able to identify the customer through the account number I_c.

4. Security of Screening

In this section we consider the security of representation screening. We first note that all other enhancements do not alter the security assumptions of previous schemes [3, 1] and therefore the same security arguments hold. As already noted, the security of the tree batched signature scheme follows from the security of Merkle's authentication tree [17, 11].

Recall that screening may be applied to batch verify a number of signatures, and in our cash system, this involves proving knowledge of the representation pairs $(x1_i, x2)$. In the following we argue that if the merchant performs a fully qualified batch verification, using a small exponents test or buckets test, then the bank can perform a representation screening operation to confirm that the customer has knowledge of the coin's representation and retains the ability to reveal customer identity when double spending occurs. We also show that if the merchant performs representation screening during payment, then it is possible for the customer to successfully double spend a coin without detection.

4.1 Bank Screening

As long as the merchant rejects false coins during payment, the bank is merely duplicating the merchant's check to ensure responses are of the correct form ($du_1s + x1_i$, $ds + x2_i$). Suppose that during deposit a double spent coin is detected. Then the bank can perform a full verification on the coin in question with both sets of deposited parameters. Recall that the merchant's identity is stored with the deposited coin and used in the re-calculated challenge value d'. There are two possibilities:

1. The response values deposited by the merchant are correct, in which case the full verification will hold for both sets of parameters. This means that the customer has double spent and its identity will be revealed along with the secret value u as proof of double spending.
2. The response values deposited by the merchant are incorrect. In this case the merchant has cheated and is trying to deposit the coin twice. Notice that this case includes the possibility that the merchant is in collusion with the customer. Since the merchant will always be caught if he colludes, there is no incentive for him to do so.

4.2 Merchant Screening

If the merchant only performs screening during payment, rather than a true verification, then it can easily be seen that the customer is able to double spend without identification. Under the assumption that the merchant performs a representation screen, of the form $\prod_{i=j}^{k} g_1^{r1_i} g_2^{r2_i} = \prod_{i=j}^{k} A^d B_i$, during payment, the customer is able to provide an alternative set of r-values that enables him to double spend a coin. For example, the customer may compute new r-values $r1_a' = (x.r1_a)$ and $r1_b' = (y.r1_b)$ and double spend the coin B_a whose representation is $r1_a = du_1s + x1_a$ and $r_{2a} = ds + x2_a$, as follows[1]. Select $x \in_R \mathbb{Z}_q$ and compute y such that: $(x \times r1_a) + (r1_b - y) = r1_a + r1_b$, (e.g. If $r1_a$=3, $r1_b$=7, and x=2, then y=3). When mixed into a batch, these values would still enable the batch representation check to succeed.

When the merchant finally deposits the coin, the bank will detect the double spent coin B_a and will see that it is the customer who has defrauded the system (due to different merchant parameter t). Now, if the customer has provided a new set of

---

[1] It is sufficient to provide a false set of r1-values, with correspondingly correct r2-values, to double spend successfully.

r-values, that satisfy the representation check, it is straightforward to see that the bank will not be able to reveal the associated identity.

5. Efficiency

We now review the efficiency of the cash scheme. We exclude the hash operations and also precomputation, assuming that the precomputation is conducted during processor idle time. We conduct an informal comparison to divisible coin based schemes, Okamoto [14] and Easy Come Easy Go cash (*EasyGo*) [5]. Applying the analysis of [5], we demonstrate greater efficiency over Okamoto's scheme and a somewhat comparative efficiency to *EasyGo*.

Comparisons to the divisible schemes are conducted on the basis that the customer withdraws 1000 coins from the bank. We borrow the analysis of [5], and set the security equivalent to a 512 bit modulus, even though today a much larger value would be desirable. Consequently, detachable coin parameters are p = 512 bits and q = 160 bits, with a 160 bit digest function. Divisible schemes employ a 512 RSA modulus N. Given that the complexity of modular exponentiation is $O(\log |q| \log^2 |p|)$ [10], we note that some of the divisible scheme's operations (namely the square roots) are approximately 3.2 (=512/160) times more expensive than our modular exponentiations. This is calculated as each square root is equivalent in effort to an exponentiation, where the exponent value is equal to N (i.e. 512 bits). It is worth noting that this difference would be increased with a larger composite modulus such as would be required today for an implementation of divisible schemes.

5.1 Detachable Coins

The increase in the size of each coin is determined by the depth of the binary tree constructed during the tree build-up stage, translating to $(k + 2r \div (2^k + r)) \times |h|$ additional bits for a tree with $2^k - 1$ leaf nodes of depth k and 2r of depth k + 1. For example, a balanced tree of depth 11 (accommodating 1024 coins) adds 10×160 bits to the signature (200 bytes). In contrast to the scheme in [1], 1023×160 bits is required for the signature residue (~20Kbytes). This represents a significant improvement over the linear increase, reducing signature size and consequently the required communication costs.

During withdrawal both the bank and customer are required to execute the protocol once for an arbitrary number of coins, the effort for the bank consists of 2 modular exponentiations, whilst the customer performs 6 modular exponentiations. During communications 188 bytes of data is transferred. In addition to a fixed 296 bytes, the customer provides storage for each withdrawn coin consisting of 64 bytes per coin.

The payment protocol involves two modular multiplications/additions per coin spent by the customer, whilst the merchant performs 4 modular exponentiations and

2 multi-exponentiations for an arbitrary number of coins received. Communications costs involve the transmission of each coin to be spent; this includes A, coin tokens B_i, response values r_i, and signature (T_{resij}, z' , a', b' r'), giving 252 + 104n bytes plus 20 log(n) signature residue bytes.

Similar to the merchant, the bank performs 4 modular exponentiations and 2 multi-exponentiations for an arbitrary number of coins received during deposit. The small exponents test introduces an additional computation to ensure that the representation check is valid. Borrowing the analysis of [2][2], a batch size of up to 200 remains efficient using the small exponents tests. However, as the batch size increases beyond this, the Bucket test becomes the more efficient. Furthermore, where screening is sufficient for the bank no additional exponentiations are required.

By extension of [1], it follows that our scheme is more efficient than Brands' [3] for the withdrawal and payment of multiple coins. We now compare against divisible based schemes, generally accepted as the most efficient paradigm for electronic cash.

5.2 Okamoto Divisible Cash

Okamoto [14] proposed the first true practical electronic cash scheme, recently improved upon by [5]. In our scheme the withdrawal of each detachable coin is realised with 2 multi-exponentiations and 4 modular exponentiations by the customer; the bank performs 2 modular exponentiations. In communication, the bank transmits about 512 bytes of data whilst the customer transmits only 128 bytes. Okamoto's scheme involves only one modular exponentiation and a 128 byte message for both the customer and bank. As suggested in [5], to accommodate unlinkability between different divisible coins an additional 4,000 exponentiations is required during each withdrawal.

The payment stage requires between 2 and 22 (average 11) exponentiations depending on the coin denomination, for both customer and merchant. As mentioned above, these exponentiations are 3.2 times more expensive, and computationally equivalent to 35 of our exponentiations. In contrast, 5 modular exponentiations and 2 multi-exponentiations are conducted by the merchant in our scheme. On average 2304 bytes are transmitted by the customer, whilst our payment involves communication of a 252 bytes + 104 bytes per coin, and log(n) signature residue bytes.

5.3 Easy Come Easy Go Divisible Cash

During withdrawal the merchant performs 2 exponentiations and the user conducts 10 exponentiations. At payment time, the merchant performs a similar initial computation to our scheme plus the addition denomination revelation phase, on

---

[2] Whilst the analysis is applied to batch signature verification, we note that this is also applicable to the batch representation check.

average contributing to an additional 11 modular square roots, equivalent to 35 of our modular exponentiations.

Our scheme may be viewed as computationally more efficient during the on-line phase of payment, whilst the size of a detachable coin is greater; that is, an additional O(log n) signature residue bytes per coin is incurred for single coin payments. However, our scheme possesses O(n) precomputation activities.

6. Conclusions

In this paper we have presented a practical electronic cash system that is not based upon the divisible coin paradigm. We applied the concepts of batch cryptography in a novel way to obtain detachability of coins for the partitioning of cash. The scheme overcomes a number of deficiencies presented in our previous batch coin scheme, including a significant reduction in coin signature size. We also analysed and discussed several additional general efficiency improvements. We conducted an informal comparison to the divisible-coin approach, noting that the efficiency of our scheme exceeds previous divisible cash schemes [14] and is comparable to other well known schemes based upon divisibility [5].

Acknowledgements

We would like to thank Khanh Quoc Nguyen for drawing our attention to the related work of Tsiounis [19].

References

1. C. Boyd, E.Foo and C. Pavlovski, Efficient Electronic Cash Using Batch Signatures, 4th Australasian Conference on Information Security and Privacy – ACISP 99, LNCS, Vol. 1587, Springer-Verlag, pp244-257, 1999.
2. M. Bellare, J. A. Garay, T. Rabin, Fast Batch Verification for Modular Exponentiation and Digital Signatures, Proceedings of Eurocrypt '98, LNCS, Vol. 1403, Springer-Verlag, 1998.
3. S. Brands, Untraceable Off-line Cash in Wallet with Observers, Advances in Cryptology - Crypto '93, LNCS, Springer-Verlag, Vol. 773, pp302-318, 1993.
4. D. Chaum, E. Evertse, J. Graaf, "An improved protocol for demonstrating possession of discrete logarithms and some generalisations", Eurocrypt '87, LNCS, Springer-Verlag, Vol 304, pp127-141, 87.
5. A. Chan, Y. Frankel, Y. Tsiounis, Easy Come - Easy Go Divisible Cash, Advances in Cryptology - Eurocrypt '98, LNCS, Springer-Verlag, Vol 1403, pp561-574, 1998.
6. JS. Coron, D. Naccache, On The Security Of RSA Screening, PKC '99, 1999.
7. D. Chaum, T. Pedersen, Wallet databases with observers, Crypto '92, pp89-105, 1992.

8. N. Ferguson, Single Term Off-Line Coins, Advances in Cryptology - Crypto '93, Springer-Verlag, pp318-328, 1994.
9. A. Fiat, Batch RSA, Crypto '89, LNCS, Vol. 435, Springer-Verlag, pp175-185, 1990.
10. N. Koblitz, A course in number theory and cryptography, Graduate Texts in Mathematics, Springer-Verlag, Vol. 114, 1987.
11. R. C. Merkle, A Certified Digital Signature, Advances in Cryptology - Crypto '89, Springer-Verlag, pp218-238, 1989.
12. D. Naccache, D. M'Raihi, D. Rapheali, S. Vaudenay, Can DSA be improved: complexity trade-offs with the digital signature standard, Proceedings of Eurocrypt '94, pp85-94, 1994.
13. K. Q. Nguyen, Y. Mu, V. Varadharajan, One-Response Off-line Digital Coins, Proceedings of SAC '97, 1997.
14. T. Okamoto, An Efficient Divisible Electronic Cash Scheme, Advances in Cryptology - Crypto '95, LNCS, Vol. 963, Springer-Verlag, pp438-451, 1995.
15. T. Okamoto, K. Ohta, Divertible Zero-Knowledge Interactive Proofs and Commutative Random Self-Reducibility, Advances in Cryptology - Eurocrypt '89, LNCS, Springer-Verlag, Vol. 34, pp481-946, 1989.
16. T. Okamoto, K. Ohta, Universal Electronic Cash, Advances in Cryptology - Eurocrypt '91, LNCS, springer-Verlag, pp324-337, 1992.
17. C. Pavlovski, C. Boyd, Efficient Batch Signature Generation Using Tree Structures, CrypTEC '99, Hong Kong, 1999.
18. CP. Schnorr, Efficient Signature Generation for Smart Cards, Advances in Cryptology - Crypto '89, Springer-Verlag, pp239-252, 1990.
19. Y. Tsiounis, Efficient Electronic Cash: New Notions and Techniques, Ph.D. Thesis, College of Computer Science, Northern University, Boston MA. June 1997.
20. Y. Yacobi, M. Beller, Batch Diffie-Hellman Key Agreement Systems and their Application to Portable Communications, Proceedings of Eurocrypt '92, Vol. 658, pp208-217, 1992.
21. S. Yen, C. Laih, Improved Digital Signature Suitable for Batch Verification, IEEE Transactions on Computers, Vol. 44, No. 7, July 1995.

Appendix A: Alternative Batch Verifiers

Bellare *et. al.* [2] introduce the notion of screening for batch verification. More specifically, they prove that a simplified batch verification does not represent a true signature verification but rather a weaker property called *screening* applies. They show that screening ensures that only the true signer could have participated in generating the purported signatures, even though the individual signatures presented may not be correct. In respect of the proposed cash system, this means the in order to derive the product Πr_i, the prover must know the representation values $x1_i$ and x2.

Several alternative batch verifiers are also introduced in [2] and may be fitted to the representation check. In our system, the security values w, introduced by Yen and Laih [21], are used to strengthen the batch representation check; in the context of [2] this is termed a batch *verifier*. This prevents the customer, with high probability, from choosing incorrect r-values that satisfy the representation check. Some alternative techniques include the *Small Exponents* (roughly the approach introduced by Yen and Laih), *Random Subset*, and the *Bucket* tests [2]. To optimise efficiency we point out that an alternative batch verifier, the Bucket test, should be used as the number of coins spent in batch increases. Adapting the Bucket test, we present a revised batch representation check, performed with a smaller exponent $|m| < |w|$.

Select security parameter $m \geq 2$, optimally chosen as in [2], set $M = 2^m$, and let $W = w/(m - 1)$. For batch size n, repeat the following W times[3]:

1. Choose $f_i \in \{1 \ldots M\}$, for $i = 1 \ldots n$.
2. For $j = 1$ to M, $F_j = \{i: f_j = j\}$, (i.e., assign the set of elements to F_j).
3. For $j = 1$ to M, $R1_j = \Sigma_{i\in Fj}\, r1_i \bmod q$, $R2_j = \Sigma_{i\in Fj}\, r2_i \bmod q$, $B' = \Pi_{i\in Fj}\, B_i$.
4. Select new values $w_1 \ldots w_M, \in \{0,1\}^m$ and confirm that the representation check holds on the following batch $(R1_1, R2_1, d, A, B'_1) \ldots (r1_M, r2_M, d, A, B'_M)$:

$$\prod_{k=1}^{M} g_1^{r1_k w_k} g_2^{r2_k w_k} = \prod_{k=1}^{M} A^{dw_k} B_k^{w_k}$$

It should be noted that the above verifier is less efficient than the small exponents test for small batches and is shown to be more efficient as the batch size grows beyond 200 in size[4]. We may assume that an optimal batch verifier from amongst these possibilities is selected by the merchant during payment depending on the number of signatures to be checked.

---

[3] Each individual test gives probability of cheating $2^{-(m-1)}$, hence we need to perform this w/(m-1) times.

[4] Although our batch operations differ from [2], we assume 200 given that our exponentiation costs are consistent in form across batch verifiers.